

版权注意事项：

- 1、书籍版权归作者和出版社所有
- 2、本PDF仅限用于个人获取知识，进行私底下的知识交流
- 3、PDF获得者不得在互联网上以任何目的进行传播
- 4、如觉得书籍内容很赞，请购买正版实体书，支持作者
- 5、请于下载PDF后24小时内删除本PDF。

架构大数据

——大数据技术及算法解析

BIG DATA

ARCHITECTING BIG DATA

——TECHNOLOGIES AND ALGORITHMS EXPLAINED

赵 勇◎编著



中国工信出版集团



电子工业出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
<http://www.phei.com.cn>

架构大数据

——大数据技术及算法解析

赵 勇 编著

電子工業出版社

Publishing House of Electronics Industry

北京 • BEIJING

内 容 简 介

本书从大数据架构的角度全面解析大数据技术及算法,探讨大数据的发展和趋势。不仅对大数据相关技术及算法做了系统性的分析和描述,梳理了大数据的技术分类,如基础架构支持、大数据采集、大数据存储、大数据处理、大数据展示及交互,还融合了大数据行业的最新技术进展和大型互联网公司的大数据架构实践,努力为读者提供一个大数据的全景画卷。

本书可作为大数据技术入门和进阶的专业书籍,同时也可作为高等院校大数据相关课程的教材和教学参考用书。

未经许可,不得以任何方式复制或抄袭本书之部分或全部内容。
版权所有,侵权必究。

图书在版编目(CIP)数据

架构大数据:大数据技术及算法解析/赵勇编著. —北京:电子工业出版社,2015.6
ISBN 978-7-121-25978-4

I. ①架… II. ①赵… III. ①互联网络—数据处理—算法分析 IV. ①TP274

中国版本图书馆 CIP 数据核字(2015)第 089418 号

责任编辑:董亚峰 特约编辑:王 纲

印 刷:北京京科印刷有限公司

装 订:北京京科印刷有限公司

出版发行:电子工业出版社

北京市海淀区万寿路 173 信箱 邮编 100036

开 本:787×1 092 1/16 印张:27.75 字数:530 千字

版 次:2015 年 6 月第 1 次印刷

印 次:2015 年 12 月第 2 次印刷

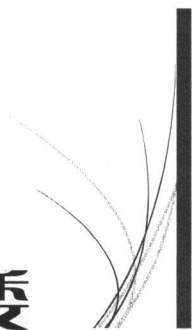
定 价:68.00 元

凡所购买电子工业出版社图书有缺损问题,请向购买书店调换。若书店售缺,请与本社发行部联系,联系及邮购电话:(010) 88254888。

质量投诉请发邮件至 zltts@phei.com.cn, 盗版侵权举报请发邮件至 dbqq@phei.com.cn。

服务热线:(010) 88258888。

编 委



李有福	陈 尧	张 迎
刘春雷	杨凡超	袁双双
孟庆祥	王心杰	查 敏

前言



大数据被称为新时代的黄金和石油，相关技术发展迅猛，所应用的行业也非常广泛，从传统行业如医疗、教育、金融、旅游，到新兴产业如电商、计算广告、可穿戴设备、机器人等。大数据技术更是国家科技发展和智慧城市建设的基石。当前“互联网+”新业态的发展，其核心也是大数据的采集、分析、价值挖潜和应用。

当今全球大数据的竞争及战略布局，突出体现在大数据的技术创新和人才培养方面。技术创新能够保障在大数据发展的浪潮中始终处于引领地位，而大数据人才体系则是能最终实现技术创新和实践的根本。政府正在设立大数据局等管理和创新机构并开放政府数据，很多大型企业开始设立首席数据官（CDO）的职位，凸显对数据驱动的经济和业务模式的重视。未来 5 年全球将有数十万名的大数据人才缺口，以及数百万名大数据的管理和支持人员缺口，因此美国的哈佛商业评论把“数据科学家”称为 21 世纪最性感的职业，全球都在开始上演大数据的人才争夺战。

我们在《大数据革命——理论、模式与技术创新》一书中对数据科学理论、大数据创新模式以及大数据技术作了初步的探索 and 介绍，在成书的过程中我觉得如果有一本更深入、更全面的技术类的大数据书籍，能够更有效地帮助读者掌握和运用大数据，这也是写这本书的初衷。由于大数据技术是当今最前沿的技术，所涉及的领域及知识跨度都非常大，发展又日新月异，所以有些内容在开始写的同时，就面临老化的危险。过去一两年的时间里大数据技术的更新令人咋舌，老牌的 Hadoop 大数据处理平台已经成熟并广泛应用在诸多行业，而很多新兴的系统和算法，如加州大学伯克利分校研究开发的 Spark 大数据通用计算平台、用于图像和视频识别的深度学习大有后来居上的势头。因

此我们也是尽量跟上形势，努力提供给读者一个大数据技术的全景画卷。

本书对大数据相关的技术及算法做了系统性的分析和描述，梳理了大数据的技术分类如基础架构支持、大数据采集、大数据存储、大数据处理、大数据展示及交互，列举了资源管理调度、数据分析和挖掘、深度学习、精准营销、社会计算等大数据相关算法，还融合了大数据行业的最新技术进展和大型互联网公司的大数据架构实践，目的是为了帮助大数据产业及技术人才全面认识和了解大数据的相关技术及算法，掌握大数据行业的最新发展动态，学习互联网巨头的大数据架构实践，进而培养大数据的技术选型和系统架构能力，成为大数据时代的创新创业先锋。

最后感谢电子科技大学极限网络计算与服务实验室的老师和同学们为本书所付出的辛勤和努力，也感谢清华大学苏州研究院大数据处理中心的行业应用实践为本书提供的指导和建议。我之前在微软的同事沈寓实和李雨航两位专家在云计算体系及安全防护方面提供了详细的资料和建议。我想向本书的责任编辑董亚峰先生和电子工业出版社的编辑们致敬，他们是本书的幕后英雄。我的妻子昆和女儿 Sophie 给予我最大的理解和支持，我爱你们。

由于我们对大数据相关技术及算法的理解和专业知识水平都有局限，本书的错误和疏漏之处在所难免，敬请各位读者谅解和指正。请将您的意见和建议发送至 dyf@phei.com.cn，感谢您的支持。

赵 勇

2015 年 5 月 12 日

目 录



第 1 章 大数据技术概述	1
1.1 大数据的概念	1
1.2 大数据的行业价值	4
1.3 大数据问题的爆发	9
1.4 大数据处理流程	12
1.5 大数据技术	13
1.5.1 基础架构支持	14
1.5.2 数据采集	14
1.5.3 数据存储	15
1.5.4 数据计算	16
1.5.5 展现与交互	18
1.6 练习题	19
参考文献	19
第 2 章 大数据基础支撑——数据中心及云计算	20
2.1 数据中心概述	20
2.1.1 云计算时代数据中心面临的问题	21
2.1.2 新一代数据中心关键技术	22
2.1.3 业界发展动态	24
2.1.4 小结	25
2.2 云计算简介	25
2.2.1 云计算定义	26
2.2.2 云计算主要特征	27

2.2.3	Web 服务、网格和云计算	28
2.2.4	云计算应用分类	29
2.2.5	小结	31
2.3	大数据与云计算的关系	32
2.3.1	大数据是信息技术发展的必然阶段	33
2.3.2	云计算等新兴信息技术正在真正地落地和实施	34
2.3.3	云计算等新兴技术是解决大数据问题的核心关键	34
2.4	云资源调度与管理	35
2.4.1	云资源管理	36
2.4.2	云资源调度策略	38
2.4.3	云计算数据中心负载均衡调度	40
2.5	开源云管理平台 OpenStack	44
2.5.1	OpenStack 的构成	45
2.5.2	OpenStack 各组件之间的关系	46
2.5.3	OpenStack 的逻辑架构	47
2.5.4	小结	49
2.6	软件定义网络	49
2.6.1	起源与发展	50
2.6.2	OpenFlow 标准和规范	50
2.6.3	OpenFlow 的应用	53
2.7	虚拟机与容器	55
2.7.1	VM 虚拟化与 Container 虚拟化	55
2.7.2	Docker	55
2.8	练习题	57
	参考文献	57
第 3 章	云计算先行者——Google 的三驾马车	59
3.1	Google 的三驾马车	59
3.1.1	GFS——一个可扩展的分布式文件系统	59
3.1.2	MapReduce——一种并行计算的编程模型	64
3.1.3	BigTable——一个分布式数据存储服务系统	69
3.2	Google 新“三驾马车”	77
3.2.1	Caffeine——基于 Percolator 的搜索索引系统	77
3.2.2	Pregel——高效的分布式图计算的计算框架	80
3.2.3	Dremel——大规模数据的交互式数据分析系统	85
3.3	练习题	89
	参考文献	89
第 4 章	云存储系统	91
4.1	云存储的基本概念	91
4.1.1	云存储结构模型	91

4.1.2	云存储与传统存储系统的区别	94
4.2	云存储关键技术	95
4.2.1	存储虚拟化技术	95
4.2.2	分布式存储技术	97
4.3	云存储系统分类	98
4.3.1	分布式文件存储	99
4.3.2	分布式块存储	105
4.3.3	分布式对象存储	109
4.3.4	统一存储	117
4.4	其他相关技术	124
4.5	练习题	127
	参考文献	127
第5章	数据采集系统	129
5.1	Flume	130
5.1.1	Flume 架构	130
5.1.2	Flume 核心组件	133
5.1.3	Flume 环境搭建与部署	134
5.2	Scribe	139
5.2.1	Scribe 架构	139
5.2.2	Scribe 中的 Store	140
5.2.3	Scribe 环境搭建与部署	141
5.3	Chukwa	144
5.3.1	Chukwa 的设计目标	144
5.3.2	Chukwa 架构	145
5.3.3	Chukwa 环境搭建与部署	147
5.4	Kafka	150
5.4.1	Kafka 架构	150
5.4.2	Kafka 存储	152
5.4.3	Kafka 的特点	152
5.4.4	Kafka 环境搭建与部署	154
5.5	练习题	155
	参考文献	155
第6章	Hadoop 与 MapReduce	156
6.1	Hadoop 平台	156
6.1.1	Hadoop 概述	156
6.1.2	Hadoop 的发展简史	157
6.1.3	Hadoop 的功能和作用	158
6.1.4	HDFS	159
6.1.5	HBase	162

6.2	MapReduce	171
6.2.1	第一代 MapReduce (MRv1)	172
6.2.2	MapReduce 2.0——Yarn	180
6.3	Hadoop 相关生态系统	184
6.3.1	交互式数据查询分析	184
6.3.2	数据收集、转换工具	187
6.3.3	机器学习工具	188
6.3.4	集群管理与监控	188
6.3.5	其他工具	189
6.4	Hadoop 应用案例	191
6.5	练习题	192
	参考文献	192
第 7 章	Spark——大数据统一计算平台	193
7.1	Spark 简介	193
7.1.1	Spark	193
7.1.2	BDAS	195
7.2	RDD	197
7.2.1	RDD 基本概念	197
7.2.2	RDD 示例	199
7.2.3	RDD 与分布式共享内存	200
7.3	Spark SQL	201
7.4	MLlib	203
7.5	GraphX	206
7.6	Spark Streaming	206
7.6.1	基本概念	207
7.6.2	编程模型	208
7.7	Spark 的安装	210
7.7.1	单机运行 Spark	210
7.7.2	使用 Spark Shell 与 Spark 交互	213
7.8	Shark、Impala、Hive 对比	214
7.9	练习题	216
	参考文献	216
第 8 章	Storm 流计算系统	218
8.1	流计算系统	218
8.1.1	流计算系统的特点	218
8.1.2	流计算处理基本流程	219
8.2	Storm 流计算框架	220
8.2.1	Storm 简介	220
8.2.2	Storm 关键术语	221

8.2.3 Storm 架构设计	223
8.3 Storm 编程实例	225
8.4 Storm 应用	228
8.4.1 Storm 应用场景	228
8.4.2 Storm 应用实例	228
8.5 其他流计算框架	229
8.6 练习题	231
参考文献	231
第 9 章 SQL、NoSQL 与 NewSQL	232
9.1 传统 SQL 数据库	232
9.1.1 关系模型	232
9.1.2 关系型数据库的优点	233
9.1.3 关系型数据库面临的问题	234
9.2 NoSQL	234
9.2.1 NoSQL 与大数据	235
9.2.2 NoSQL 理论基础	235
9.2.3 分布式模型	238
9.2.4 NoSQL 数据库分类	241
9.3 NewSQL	255
9.3.1 系统分类	255
9.3.2 Google Spanner	256
9.3.3 MemSQL	258
9.3.4 VoltDB	260
9.4 练习题	263
参考文献	263
第 10 章 大数据与数据挖掘	264
10.1 数据挖掘的主要功能和常用算法	264
10.1.1 数据挖掘的主要功能	264
10.1.2 常用算法	265
10.2 大数据时代的数据挖掘	280
10.2.1 传统数据挖掘解决方案	280
10.2.2 分布式数据挖掘解决方案	280
10.3 数据挖掘相关工具	282
10.3.1 Mahout	282
10.3.2 语言工具——Python	288
10.4 数据挖掘与 R 语言	289
10.4.1 R 语言简介	289
10.4.2 R 语言在数据挖掘上的应用	290
10.5 练习题	294

参考文献	294
第 11 章 深度学习	298
11.1 深度学习介绍	299
11.1.1 深度学习的概念	299
11.1.2 深度学习的结构	299
11.1.3 从机器学习到深度学习	301
11.2 深度学习基本方法	302
11.2.1 自动编码器	302
11.2.2 稀疏编码	304
11.3 深度学习模型	305
11.3.1 深度置信网络	306
11.3.2 卷积神经网络	308
11.4 深度学习的训练加速	310
11.4.1 GPU 加速	310
11.4.2 数据并行	311
11.4.3 模型并行	312
11.4.4 计算集群	313
11.5 深度学习应用	313
11.5.1 Google	314
11.5.2 百度	314
11.5.3 腾讯 Mariana	315
11.6 练习题	316
参考文献	316
第 12 章 电子商务与社会化网络大数据分析	318
12.1 推荐系统简介	318
12.1.1 推荐系统的评判标准	319
12.1.2 推荐系统的分类	319
12.1.3 在线推荐系统常用算法介绍	320
12.1.4 相关算法知识	323
12.2 计算广告	327
12.2.1 计算广告简介	327
12.2.2 计算广告发展阶段	327
12.2.3 计算广告相关算法	330
12.2.4 计算广告与大数据	332
12.2.5 大数据在计算广告中的应用案例	333
12.3 社交网络	333
12.3.1 社交网络中大数据挖掘的应用场景	334
12.3.2 社交网络大数据挖掘核心算法模型	334
12.3.3 图计算框架	335

12.3.4 大数据在社交网络中的应用案例	337
12.4 练习题	338
第 13 章 大数据展示与交互技术	339
13.1 数据可视化分类	339
13.1.1 按照展示内容进行划分	340
13.1.2 按照数据类型进行划分	341
13.2 可视化技术分类	351
13.2.1 2D 展示技术	351
13.2.2 3D 渲染技术	356
13.2.3 体感互动技术	360
13.2.4 虚拟现实技术	362
13.2.5 增强现实技术	364
13.2.6 可穿戴技术	365
13.2.7 可植入设备	368
13.3 练习题	369
参考文献	369
第 14 章 大数据安全与隐私	372
14.1 云计算时代安全与隐私问题凸显	372
14.2 云计算与大数据时代的安全挑战	374
14.2.1 大数据时代的安全需求	374
14.2.2 信息安全的发展历程	375
14.2.3 新兴信息技术带来的安全挑战	376
14.3 如何解决安全问题	380
14.3.1 云计算安全防护框架	381
14.3.2 基础云安全防护关键技术	384
14.3.3 创立本质安全的新型 IT 体系	387
14.4 隐私问题	389
14.4.1 防不胜防的隐私泄露	389
14.4.2 隐私保护的政策法规	390
14.4.3 隐私保护技术	391
14.5 练习题	393
参考文献	393
第 15 章 大数据技术发展趋势	394
15.1 实时化	394
15.2 内存计算	396
15.2.1 机遇与挑战	396
15.2.2 研究进展	397
15.2.3 发展展望	399

15.3 泛在化	399
15.3.1 发展现状	400
15.3.2 发展趋势	401
15.4 智能化	406
15.4.1 传统人工智能	406
15.4.2 基于大数据的人工智能	407
15.5 练习题	410
参考文献	410
第 16 章 知名企业大数据架构简介	411
16.1 腾讯	411
16.1.1 背景介绍	411
16.1.2 整体架构	412
16.2 淘宝	416
16.2.1 背景介绍	416
16.2.2 整体架构	416
16.3 Facebook	417
16.3.1 背景介绍	417
16.3.2 整体架构	418
16.3.3 技术架构展望	420
16.4 Twitter	420
16.4.1 背景介绍	420
16.4.2 整体架构	420
16.4.3 技术架构展望	422
16.5 Netflix	422
16.5.1 背景介绍	422
16.5.2 整体架构	423
16.5.3 Netflix 个性化和推荐系统架构	426
16.6 练习题	430
参考文献	430

第1章

大数据技术概述

近年来，以物联网、移动互联网、云计算和大数据（Big Data）为代表的新一代信息技术发展迅猛，而大数据则风头最劲。无所不在的移动终端、智能设备、无线传感器等每秒都在产生数据，拥有数以亿计用户的互联网服务时时刻刻在产生巨量的交互，比如，百度每天大约要处理几十 PB 的数据，Twitter 每天会产生 7TB 的数据，Facebook 每天生成 300TB 以上的日志数据等。数据产生的速度太快，要处理的数据量也太大。据 IDC 预测，到 2020 年全球将拥有 35ZB（1ZB = 10^{21} 字节）的数据。与此同时，数据的价值也在不断凸显，数据被类比为新时代的黄金和石油，现代企业快节奏的业务需求和竞争压力对数据处理的实时性和有效性提出了更高的要求，传统的数据处理技术已经完全不能满足大量数据的实时处理的需求，大数据就全面爆发了。大数据涉及国家战略、区域及企业发展、社会民生的方方面面，掌握大数据的核心理念、模式和技术，就把握了新时代的脉搏。2013 年被称为大数据元年，在 2014 年，大数据不管在技术还是应用方面都取得了很大的发展。本书主要介绍大数据技术及算法的相关知识及最新进展，帮助大数据从业人士了解、掌握和架构大数据。

在本章中，我们将重点介绍大数据的相关概念、历史发展、大数据的价值、大数据问题、大数据处理流程以及大数据技术分类等几个方面。

1.1 大数据的概念

大数据指的是无法在规定时间内用现有的常规软件工具对其内容进行抓取、管理和处

理的数据集合。大数据技术则特指新一代的创新型的技术，能够突破常规软件的限制，是对大数据进行采集、存储和处理的技术的统称。

大数据（Big Data）一词正式出现是在 2011 年麦肯锡全球研究院发布的《大数据：下一个创新、竞争和生产力的前沿》研究报告中。但其实从 20 世纪 90 年代至 21 世纪初，大数据已经开始萌芽。那时对于大数据的研究主要集中在算法、模型、模式、识别等几个方面，数据挖掘理论和数据库技术也在趋于成熟，一批数据仓库、知识管理系统等商业智能工具逐步出现并被应用。这些方面的研究为大数据时代的到来奠定了良好的理论基础。

从 2003 年开始，业内学者开始围绕着半结构化数据或非结构化数据的处理进行探索，大数据的发展也取得了一定的突破。传统的数据处理技术能够较好地处理结构化的数据，而对非结构化的数据则难以应对。对非结构化数据的处理带动了数据处理技术的发展。2005 年 Hadoop 项目诞生，由多个软件产品组成的 Hadoop 生态系统为结构化和复杂数据的处理提供了一个快速、可靠分析的平台。其中分布式文件系统 HDFS 的可靠数据存储服务和高性能的 MapReduce 并行数据处理服务为大数据的处理提供了良好的技术基础。

2006—2009 年，并行计算和分布式系统成为主流，大数据技术的发展进入成熟期。从 2009 年开始，各国政府对于大数据技术的重视和应用已经初现端倪。在 2009 年，联合国启动了“全球脉动”（Global Pulse）计划，旨在推动数字数据快速收集和分析方式的创新；美国启动 Data.gov 网站，向公众开放各种政府数据；欧洲一些领先的研究型图书馆和科技信息研究机构开始合作，从而改善在互联网上获取科学数据的简易性。

2010 年以来，智能手机的应用日益广泛，随之而来的数据的碎片化、分布式、流媒体等特征更加明显，移动数据量急速增长。2011 年 5 月，麦肯锡全球研究院发布的《大数据：下一个创新、竞争和生产力的前沿》研究报告中指出，大数据已经渗透到当今的每一个行业和业务职能领域成为重要的生产因素。报告中提出了可能改变世界格局的 12 项技术，包括云计算、物联网、移动互联网、知识工作自动化、先进机器人等，而大数据则是这些技术的基础，每项技术都离不开大数据。同年 12 月，信息处理技术作为 4 项关键技术之一在我国的十二五规划上被提出来，其中包括海量数据存储、数据挖掘、图像视频智能分析等技术，而这些都是大数据的重要组成部分。另外 3 项关键技术，包括信息感知技术、信息传输技术、信息安全技术，也都与大数据密切相关。

2012 年 1 月份，在瑞士达沃斯召开的世界经济论坛上，大数据是主题之一，会上发布的报告《大数据，大影响》宣称，数据像货币或黄金一样，已经成为一种新的经济资产类别。2012 年 3 月，美国政府发布了《大数据研究和发展倡议》，并划拨 2 亿美元的专项支持资金，标志着大数据已经成为重要的时代特征。美国政府对数据领域的投资，也使得大数据的应用从一开始单纯的商业行为上升到国家科技战略，引发了全球对大数据的关注和追捧。

2012 年 4 月，美国的大数据处理公司 Splunk 成功上市，这也是第一家上市的大数据处理公司。该公司提供大数据监测和分析服务的软件，其成功上市也促使其他 IT 公司加快对大数据技术和应用的布局。7 月，阿里巴巴推出数据分享平台“聚石塔”，为淘宝、天猫

上的电商和电商服务商提供数据云服务,希望通过分享和挖掘海量数据,为国家和中小企业提供价值。阿里巴巴集团也成为国内最早提出企业数据化运营的企业。

2012年7月,联合国在纽约发布了一份关于大数据政务的白皮书,总结了各国政府如何利用大数据更好地服务和保护人民。这份白皮书举例说明在一个数据生态系统中,个人、公共部门和私人部门各自的角色、动机和需求。例如通过对价格关注和更好服务的渴望,个人提供数据和众包信息,并对隐私和退出权力提出需求;公共部门出于改善服务、提升效益的目的,提供统计数据、设备信息、健康指标及税务和消费信息等,并对隐私和退出权力提出需求;私人部门出于提升客户认知和预测趋势目的,提供汇总数据、消费和使用信息,并对敏感数据所有权和商业模式更加关注。联合国还以爱尔兰和美国的社交网络活跃度增长可以作为失业率上升的早期征兆为例,表明政府如果能合理分析所掌握的数据资源,将能“与数俱进”,快速应变。

2013年称为大数据元年,几乎所有的大型互联网企业都将业务范围延伸至大数据产业。电子商务平台、社交平台、门户网站等,都存在着大数据的影子。大数据也由技术热词逐渐演变成社会浪潮,影响着国家、社会和生活的各个方面。

全球大数据产业的日趋活跃,技术演进和应用创新的加速发展,使各国政府逐渐认识到大数据在推动经济发展,改善公共服务,增进人民福祉,乃至保障国家安全方面的重大意义。2014年5月,美国白宫发布了2014年全球“大数据”白皮书的研究报告《大数据:抓住机遇、守护价值》。报告鼓励使用数据以推动社会进步,特别是在市场与现有的机构并未以其他方式来支持这种进步的领域;同时,也需要相应的框架、结构与研究,来帮助保护美国人对于保护个人隐私、确保公平或防止歧视的坚定信仰。

近年来大数据不断地向社会各行各业渗透,已经开始广泛应用于教育、金融、医疗等各个行业,使得大数据的技术领域和行业边界愈来愈模糊不定,应用创新已超越技术本身更受到青睐。大数据技术可以为每一个领域带来变革性影响,并且正在成为各行各业颠覆性创新的原动力和助推器。

大数据需要满足数据量足够大(Volume)、数据的种类多样(Variety)、数据的增长及处理速度快(Velocity)、数据蕴藏价值大(Value)这4个根本特征,才能称之为大数据。

数据量大(Volume)指的是数据的采集、存储和计算的量都非常大,大数据通常指10TB以上规模的数据量。造成数据量增大的原因有很多,例如,很多监控和传感设备的使用,使我们感知到更多的事务,这些事务的数据将被部分或者完全存储;(移动)通信设备的使用,使得交流的数据量成倍增长;基于互联网和社会化网络的应用的发展,数以亿计的用户每天产生大量的数据。

数据种类多(Variety)是指数据的种类和来源较多,例如多种传感器、智能设备、社交网络等。数据的种类包括结构化、半结构化和非结构化数据,包括图片、音频、视频、地理位置等多类型的数据。

数据的增长及处理速度快(Velocity)指数据每分每秒都在爆炸性地增长,而对数据的

处理速度要求也很高，数据的快速动态的变化使得流式数据成为大数据的重要特征，对大数据的处理要求具有较强的时效性，能够实时地查询、分析、推荐等。

数据的价值大（Value）是指在海量的数据中，存在着巨大的被挖掘的商业价值，然而由于数据总量的不断增加，数据的单位价值密度却相对较低，如何通过强大的数据挖掘算法，结合企业的业务逻辑来从海量数据中获取有用的价值是大数据要解决的重要问题。

除了上述的4个主要特征外，大数据与传统的数据处理技术最明显的一个特征区别是，大数据的处理要求是在线的。例如，用户在使用某一网站或应用时，需要及时地把用户行为数据传送给企业，通过相应的数据处理或数据挖掘算法，分析出用户的行为特征，并根据处理结果对用户进行精准的内容推荐或行为预测，在提升用户体验的同时，增加用户黏度，为企业带来更多的商业价值。而离线的数据处理，则不能满足这一需求，在线实时处理也是大数据发展的重要趋势和特点。

1.2 大数据的行业价值

大数据在过去几年得到了全社会的关注和快速的发展，几乎在每个行业都可以见到大数据应用的影子。大数据的应用范围越来越广，应用的行业也越来越多，我们几乎每天都可以看到大数据的一些新奇应用，大数据的价值也已经体现在方方面面。大数据目前较多的应用领域主要有互联网、金融、医疗、教育、政府等行业，应用的环境也不尽相同，下面介绍几种大数据的典型应用场景。

1. 分析用户行为，建立数据模型，并进行预测

大数据在用户行为分析和预测方面的应用是最突出的。企业通过对用户社交网站的行为数据、浏览器的日志信息、传感器的数据等进行收集和分析，就可以得到用户的行为习惯，通过建立出数据模型，可以对用户的下一步行为进行预测。

在用户的行为分析方面，最经典的案例应该是美国沃尔玛公司（WalMart）将尿不湿和啤酒摆放在一起的销售策略。沃尔玛对顾客的购物习惯进行关联规则分析，从中得出顾客会经常一起购买哪些商品。沃尔玛利用数据挖掘工具对其保存在数据仓库里面的所有门店的交易数据进行分析，得出了和尿不湿一起购买最多的商品是啤酒的结论。沃尔玛在所有的门店里将尿不湿与啤酒并排摆放在一起，结果是尿不湿与啤酒的销售量双双增长。

另外一个比较著名的例子就是 Target 怀孕预测的案例。他们对商品数据库里的数万类商品和女性顾客的商品购买记录进行分析，挖掘出与怀孕高度相关的25项商品，制作“怀孕预测”指数，可以精确地预测到客户在什么时候想要小孩，推算出孕妇的预产期等，从而抢先一步给女性推荐相关的产品。

在用户行为预测方面，也有不少成功案例。例如美国统计学家内特·西尔弗建立统计模型，成功预测了2012年美国大选的结果。通过他的预测，看到奥巴马有431种胜利途径，

对比罗姆尼仅有 76 种，奥巴马总统连任的机会是 86.3%。在其他行业，电信可以通过大数据预测用户的流失，从而可以提前采取相应的手段留住客户；汽车保险行业可以了解客户的驾驶水平和需求，来为顾客推荐合适的保险等。大数据对于当代企业能够更好地运营所体现出的价值已经不言而喻。

2. 提升企业的资产管理，优化企业的业务流程

大数据也可以帮助企业提升资产管理和优化业务流程。企业利用实时数据能够实现预测性的维护并减少故障，推动产品和服务开发。比如在交通和物流领域，大数据最广泛的应用就是供应链以及配送路线的优化。通过结合传感器数据，以及社交媒体、网络搜索以及天气预报数据，可以挖掘出有价值的信息。利用地理定位和无线电频率的识别追踪货物和送货车，利用实时交通路线数据制定更加优化的路线。

UPS 快递高效地利用了地理定位数据。为了使总部能在车辆出现晚点的时候跟踪到车辆的位置和预防引擎故障，它的货车上装有传感器、无线适配器和 GPS。同时，这些设备也方便了公司监督、管理员工并优化行车线路。UPS 为货车定制的最佳行车路径是根据过去的行车经验总结而来的。2011 年，UPS 的驾驶员少跑了近 4828 万千米的路程。

DHL 是全球知名的邮递和物流公司。它是一家传统行业的企业，然而在移动互联网和大数据浪潮中却并不落后，在瑞典推出了众包模式送货的移动应用 MyWays，人们可以通过移动应用报名投递自己行动路线附近的包裹，并获取报酬。此外，DHL 还把大数据应用于管理物流风险，从而为客户提供更好的服务。

3. 大数据服务智慧城市、智慧交通

智慧城市是当前我国城镇化改革的建设重点，大数据技术是实现智慧城市的核心支撑技术。智慧城市就是运用信息和通信技术手段感测、分析、整合城市运行核心系统的各项关键信息，从而对包括政务、民生、社会化管理、企业发展在内的各种需求做出智能响应。其实质是利用先进的信息技术，实现城市智慧式管理和运行，进而为城市中的人创造更美好的生活，促进城市的和谐、可持续成长。目前，在国内外，每天都会涌现出新的大数据智慧城市的应用案例。我们选取几个有代表性的案例。

随着智能电网的提出，智能电表得到了极大的普及，目前全国范围内至少有 1 亿块智能电表在使用，不仅极大地方便了普通用电用户，而且电力公司也因此收集了大量的用电数据。这些海量数据在日积月累的过程中逐渐给用电信息采集系统带来了存储和计算的压力，而且随着业务的不断深化，智能电表历经多次升级换代，采集项数翻了几倍，采集频率也逐步从一天一次向 15 分钟一次（96 次/天）升级。以一个用电用户超过 2000 万户的省公司来说，一天的数据入库量接近 20 亿次，再加上实时统计分析的要求，原有系统基于传统关系型数据库的架构已无力支撑。在这种情况下，该省公司基于清华大数据处理中心的以 Hadoop 为基础的 HBase 解决方案进行用电数据的存储和结果查询，使用 Hive 进行相关的统计分析。经过业务梳理，选择了 3 个计算场景和一个查询场景进行尝试。通过实际业务数据的计算对比，3 个计算场景用时比现有系统快 10~20 倍，查询场景的响应时间则缩

短了两个数量级，而整体集群的硬件造价仅为现有系统的 1/6，并且还具备极佳的横向扩展能力。

法国里昂市与 IBM 的研究者合作开发出能够缓解道路拥堵的系统方案。IBM 为里昂开发的系统名为 Decision Support System Optimizer（决策支持系统优化器），可以基于实时的交通情况报告来侦测和预测交通拥堵。当交管人员发现某地即将发生交通拥堵，可以及时调整信号灯让车流以最高效率运行。这个系统对于突发事件也很有用，例如帮助救护车尽快到达医院。而且随着运行时间的积累，这套系统还能够“学习”过去的成功处置方案，并运用到未来预测中。

SpotHero 是预订停车位的一个移动应用，它的网站和移动应用可以较好地解决司机找不到停车位的问题。SpotHero 能够实时跟踪停车位数据变化，打开 SpotHero，将会显示附近可用的停车位的公交车和价格，同时提供导航服务，并且可以使用预付费来占领未被使用的停车位。目前，已经能够实时监控包括华盛顿、纽约、芝加哥、巴尔的摩、波士顿、密尔沃基和纽瓦克七个城市的停车位。

4. 变革公共医疗卫生，对疾病进行预测

谷歌的 FluTrend 可以利用搜索关键词和大数据技术成功预测流感的散布趋势。在流感爆发前，人们用谷歌搜索流感的相关资讯或措施的比例将会增加，谷歌通过对无数流感关键词进行分析，可以准确快速地预测流感将在哪里出现，以及流感的散布范围。这一项目的成功也刮起了大数据变革公共健康的浪潮。目前，谷歌又孵化了一个医疗健康项目，名为 Baseline，它主要用大数据来预防癌症。

百度公司也在疾病预测方面做了一些工作。2014 年 7 月，在百度推出世界杯预测之后，又上线了一个最新服务：疾病预测。它能为用户提供流感、肝炎、肺结核和性病 4 种疾病的趋势预测，并可依据过去 30 天的资料，对未来 7 天疾病变化进行预测。目前该服务已经涵盖了我国 331 个城市，2870 个区县，并且某些城市已经细化到商圈为目标单位，未来甚至可以细化到个人的粒度。

对于目前正在暴发的埃博拉病毒，也可以通过大数据技术来预防疾病的传播，对疫情进行更好的控制，做好民众的救助工作。首先，西非等地的跨国电信业者与世界卫生组织合作，提供当地居民行为通信资料，通过分析绘制当地居民聚落位置和人口移动地图，来预测病毒散布的位置。其次，非洲政府可以根据用户的手机定位，分析出当地居住区位置的移动轨迹，规划医疗救助站的位置，从而安排最佳的救助路线，使居民远离疫情较为严重的区域。

除了在疾病预测方面，利用大数据的计算和分析能力，能够让我们在几分钟内解码整个 DNA，制定出最新的治疗方案。大数据技术目前已经在医院应用监视早产婴儿和患病婴儿的情况，通过记录和分析婴儿的心跳，医生针对婴儿的身体可能会出现不适症状做出预测，这样可以帮助医生更好地救助婴儿。

大数据已经在医疗和健康领域取得了一定的成果，将疾病防治关口前移，可以大大节

省医疗资源的消耗。有效的数据分析也可以提前对民众进行医疗健康知识的普及教育，从而较好地预防疾病的发生。

5. 在金融行业利用大数据进行战略决策和精准营销

银行、证券和保险是金融类企业的3个重要部分。国内不少银行已经开始尝试通过大数据来驱动业务运营。例如民生银行，其80%以上的客户是小微企业。借助大数据平台，民生银行的每家小微企业客户的信息都能够实时上报民生的“数据加工厂”，并生产出有价值的信息，使总行能够更加快速、准确地获得各个行业的市场需求信息，从而快速、精确地进行战略决策和市场规模。

基于大数据平台，民生银行实现了内部管理的精细化，“用数据说话、靠数据决策”已经成为民生银行的一种管理文化。依据大数据平台和专业金融技术工具，民生银行目前能够准确计算出每位客户的利润贡献度，从而真正做到个性化定价和个性化服务。在产品定价方面，以往银行都按照批量定价模式，向客户销售贷款；而个性化定价，则根据客户的存款、贷款、业务经营情况等综合指标进行科学定价，不仅能够吸引优质客户，提高客户黏性，降低客户流失率，也能够提高整体收益。基于大数据平台，民生银行实现了从“广撒网”到“批量定向开发”的转变。除了民生银行，光大银行建立了社交网络信息数据库，招商银行利用大数据发展小微贷款，中信银行信用卡中心使用大数据技术实现了实时营销。

在证券行业，大数据主要包含几个方面的应用：股价预测、客户关系管理和投资景气指数。

现在很多股权的交易都利用大数据算法进行，这些算法现在越来越多地考虑了社交媒体和网站新闻来决定在未来几秒内是买入还是卖出。IBM日本的新系统仅用6小时就预测出分析师需要花费数日才能计算出的预测值，它结合其他相关经济数据的历史数据分析与股价的关系，从而得出预测结果。

对客户关系的管理包括两个方面，对客户进行细分和客户流失的预测。通过对客户的账户状态进行分析，对客户进行聚类 and 细分，从而发现客户交易，找出最有价值和盈利潜力的客户群，为他们提供个性化服务。证券公司通过对客户的历史交易行为和流失情况进行分析，建立客户流失模型，从而预测客户流失。

在保险行业，大数据应用也包括3个方面：客户细分及精细化营销、欺诈行为分析和精细化运营。例如友邦保险使用了大数据魔镜软件，开发出客户挖掘、精准投放、二次开发、战略指导、全民分析等多种智能分析模型，为管理层提供最直接的数据依据，之前每个保险业务员从200个电话中，可能才能挖掘出两三个意向客户，而精准的投放使得平均拨打一个电话就可以得到一个客户。

6. 利用大数据保障公共安全

大数据的应用和发展可以帮助公共服务更好地优化模式，提升社会安全保障能力和面对突发情况的应急能力。作为大数据方面的开拓者——美国，在应用大数据来治理社会和稳定社会这方面的成绩显著。

美国国家安全和交通安全局基于数据挖掘技术，开发了计算机辅助乘客筛选系统，为美国本土各个机场提供应用接口。该系统将乘客购买机票时提供的姓名、联系地址、电话号码、出生日期等信息输入商用数据库中，商用数据库则据此将隐含特殊危险等级的数字分值传送给交通安全局：绿色分值的乘客将接受正常筛选，黄色分值的乘客将接受额外筛选，红色分值的乘客将被禁止登机，且有可能受到法律强制性的关照。

同时，利用大数据也可预防犯罪案件的发生。加利福尼亚州桑塔克鲁兹市使用犯罪预测系统，对可能出现犯罪的重点区域、重要时段进行预测，并安排巡警巡逻。在所预测的犯罪事件中，有 2/3 真的发生。系统投入使用一年后，该市入室行窃减少了 11%，偷车减少了 8%，抓捕率上升了 56%。

另外，大数据也可以推进案件的侦破。这方面最经典的案例应该是波士顿连环爆炸的成功告破。2013 年 4 月 15 日，美国波士顿在举办马拉松比赛的过程中发生连续炸弹爆炸案，导致 3 人死亡、183 人受伤。案件发生后警方不仅走访了事发地点附近 12 个街区的居民，收集可能存在的各种私人录像和照片，还大量收集网上信息，包括信息社交网站上出现的相关照片、录像等，并在这些网站上向公众提出收集相关信息的请求。通过对各方面数据的比对、查找，警方从录像中截取出嫌疑人照片并发出通缉令，从而为最终追捕罪犯提供了确凿的证据和可靠的参考。

7. 利用大数据促进教育行业变革

在教育工作中，特别是学校教育，数据成为教学改进显著的目标。美国国家教育统计中心已经把中小学和大学的学生学习行为、考试分数和职业规划等重要的数据存储起来，用于统计和分析。而近年来越来越多的网络在线教育和大规模开放式网络课程的兴起，使教育领域中的大数据获得了更为广阔的应用空间。

教育领域中大数据分析的最终目的是提高学生的学习成绩。美国教育部门创造了一套“学习分析系统”，将教育和大数据相结合。该系统是一个数据挖掘和案例运用的联合框架，主要向教育工作者提供影响学习成绩的原因等信息，为教师提供提高学生成绩更准确有效的办法。

美国已经存在一些企业成功地商业化运作了教育中的大数据。例如，IBM 与亚拉巴马州的莫白儿县公共学区在大数据方面展开合作，从而较好地改善了该学区的辍学情况；希维塔斯学习（Civitsa Learnig）在高等教育领域建立了最大跨校学习数据库，通过这些海量数据，可以看到学生的分数、出勤率、辍学率和保留率等数据的主要趋势；梦盒学习（Dream Box Learning）公司和纽顿（Knewton）公司已经成功创造并发布了各自的利用大数据的适应性学习系统。

在我国，百度推出了“百度预测”，在 2014 年也通过数据分析，预测出高考作文题目的出题范围将会在“生命的多彩”、“时间的馈赠”等六个领域中，并且给出了各领域命中的精确概率。对试题的精确预测，也可以较大程度上提高学生的学习成绩。

8. 大数据在改善着每个人的生活

大数据不仅应用在政府、企业，对于生活中的每个人都有较大的影响。例如，用户之前在电子商务网站想要购买某样东西的时候，需要从海量的购物列表里面找到自己喜欢的商品。电商网站能通过用户的性别、年龄、购物偏好、职业、收入、生活习惯，对用户的浏览内容进行记录，分析到用户对物品、价格等的需求，向用户推荐相应的物品，可以节省用户时间，提高交易成功率。

人们一般通过电视或者智能手机接收天气预警。而目前全球人口高达 70 亿，据 WeatherBug 应用开发商 Earth Networks 称，在非洲、南美洲和亚洲等一些欠发达地区，仍有将近 60 亿人不能在恶劣天气到来前接到预警。因此该公司利用遍布全球的数十万个传感器，监测温度、风力和雷电的变化情况，给用户提供领先的恶劣天气分析及预警。

一些婚恋网站一直都会进行各种各样的数字统计，例如全国有多少单身男女，单身比例，每个地方的男生（女生）喜欢什么样的女生（男生），不同年龄段的单身女生又会喜欢什么样的男生等。百合网独创了“心灵匹配测评系统”，系统里面涉及 30 多个维度，再加上实名认证，从而发现两个异性之间在生活习惯、价值观、兴趣爱好等各方面的契合度，从而形成高效率的精准速配，用户也可以通过百合网的分析数据来找到属于自己的合适的对象。

1.3 大数据问题的爆发

我们看到，传统的 IT 基础架构和数据管理分析方法已经不能适应大数据的快速增长，大数据的爆发是我们在信息化和社会发展中遇到的棘手问题，需要我们采用新的数据管理模式，研究和新一代的信息技术才能解决。我们把大数据问题归纳为 7 类。

1. 速度方面的问题

传统的关系型数据库管理系统（RDBMS）一般都是集中式的存储和处理，没有采用分布式架构，在很多大型企业中的配置往往都基于 IOE（IBM 服务器，Oracle 数据库，EMC 存储）。在这种典型配置中单台服务器的配置通常都很高，可以多达几十个 CPU，内存也能达到上百 GB，数据库的存储放在高速大容量的磁盘阵列上，存储空间可达 TB 级。这种配置对于传统的管理信息系统（MIS）需求来说是满足需求的，然而面对不断增长的数据量和动态数据使用场景，这种集中式的处理方式就日益成为瓶颈，尤其是在速度响应方面捉襟见肘。在面对大数据量的导入导出、统计分析、检索查询方面，由于依赖于集中式的数据存储和索引，性能随着数据量的增长而急速下降，对于需要实时响应的统计及查询场景更是无能为力。比如在物联网中，传感器的数据可以多达几十亿条，对这些数据需要进行实时入库、查询及分析，传统的 RDBMS 就不再适合应用需求了。

2. 种类及架构问题

RDMBS 对于结构化的、固定模式的数据,已经形成了相当成熟的存储、查询、统计处理方式。随着物联网、互联网以及移动通信网络的飞速发展,数据的格式及种类在不断变化和发展。在智能交通领域,所涉及的数据可能包含文本、日志、图片、视频、矢量地图等来自不同数据采集监控源的、不同种类的数据。这些数据的格式通常都不是固定的,如果采用结构化的存储模式将很难应对不断变化的需求。因此对于这些种类各异的多源异构数据,需要采用不同的数据和存储处理模式,结合结构化和非结构化数据存储。在整体的数据管理模式和架构上,也需要采用新型的分布式文件系统及分布式 NoSQL 数据库架构,才能适应大数据量及变化的结构。

3. 体量及灵活性问题

如前所述,大数据由于总体的体量巨大,采用集中式的存储,在速度、响应方面都存在问题。当数据量越来越大,并发读、写量也越来越大时,集中式的文件系统或单数据库操作将成为致命的性能瓶颈,毕竟单台计算机的承受压力是有限的。我们可以采用线性扩展的架构和方式,把数据的压力分散到很多台计算机上,直到可以承受,这样就可以根据数据量和并发量来动态增加和减少文件或数据库服务器,实现线性扩展。

在数据的存储方面,需要采用分布式可扩展的架构,比如大家所熟知的 Hadoop 文件系统和 HBase 数据库。同时在数据的处理方面,也需要采用分布式的架构,把数据处理任务,分配到很多计算节点上,同时还须考虑数据存放节点和计算节点之间的位置相关性。在计算领域中,资源分配、任务分配实际上是一个任务调度问题。其主要任务是根据当前集群中各个节点上面的资源(包括 CPU、内存、存储空间和网络资源等)的占用情况,和各个用户作业服务质量要求,在资源和作业或者任务之间做出最优的匹配。由于用户对作业服务质量的要求是多样化的,同时资源的状态也在不断变化,因此,为分布式数据处理找到合适的资源是一个动态调度问题。

4. 成本问题

集中式的数据存储和处理,在硬件、软件选型时,基本采用的方式都是配置相当高的大型机或小型机服务器,以及访问速度快、保障性高的磁盘阵列,来保障数据处理性能。这些硬件设备都非常昂贵,动辄高达数百万元,同时软件也经常是国外大厂商如 Oracle、IBM、SAP、微软等的产品,对于服务器及数据库的维护也需要专业技术人员,投入及运维成本很高。在面对海量数据处理的挑战时,这些厂商也推出了形似庞然大物的“一体机”解决方案,如 Oracle 的 Exadata、SAP 的 HANA 等,通过把多服务器、大规模内存、闪存、高速网络等硬件进行堆叠,来缓解数据压力,然而在硬件成本上,更是大幅跳高,一般的企业很难承受。

新型的分布式存储架构、分布式数据库如 HDFS、HBase、Cassandra、MongoDB 等由于大多采用去中心化的、海量并行处理 MPP 架构,在数据处理上不存在集中处理和汇总的

瓶颈，同时具备线性扩展能力，能有效地应对大数据的存储和处理问题。在软件架构上也实现了一些自管理、自恢复的机制，以面对大规模节点中容易出现的偶发故障，保障系统整体的健壮性。因此对每个节点的硬件配置，要求并不高，甚至可以使用普通的 PC 作为服务器，在服务器成本上可以大大节省，在软件方面开源软件也占据非常大的价格优势。

当然，在谈及成本问题时，我们不能简单地进行硬件、软件的成本对比。要把原有的系统及应用迁移到新的分布式架构上，从底层平台到上层应用都需要做很大的调整。尤其是在数据库模式以及应用编程接口方面，新型的 NoSQL 数据库与原来的 RDBMS 存在较大的差别，企业需要评估迁移及开发成本、周期及风险。除此之外，还须考虑服务、培训、运维方面的成本。但在总体趋势上，随着这些新型数据架构及产品的逐渐成熟与完善，以及一些商业运营公司基于开源基础为企业提供专业的数据库开发及咨询服务，新型的分布式、可扩展数据库模式必将在大数据浪潮中胜出，从成本到性能方面完胜传统的集中式大机模式。

5. 价值挖掘问题

大数据由于体量巨大，同时又在不断增长，因此单位数据的价值密度在不断降低。但同时大数据的整体价值在不断提高，大数据被类比为石油和黄金，因此从中可以发掘巨大的商业价值。要从海量数据中找到潜藏的模式，需要进行深度的数据挖掘和分析。大数据挖掘与传统的数据挖掘模式也存在较大的区别：传统的数据挖掘一般数据量较小，算法相对复杂，收敛速度慢。然而大数据的数据量巨大，在数据的存储、清洗、ETL（抽取、转换、加载）方面都需要能够应对大数据量的需求和挑战，在很大程度上需要采用分布式并行处理的方式，比如 Google、微软的搜索引擎，在对用户的搜索日志进行归档存储时，就需要多达几百台甚至上千台服务器同步工作，才能应付全球上亿用户的搜索行为。同时，在对数据进行挖掘时，也需要改造传统数据挖掘算法以及底层处理架构，同样采用并行处理的方式才能对海量数据进行快速计算分析。Apache 的 Mahout 项目就提供了一系列数据挖掘算法的并行实现。在很多应用场景中，甚至需要挖掘的结果能够实时反馈回来，这对系统提出了很大的挑战，因为数据挖掘算法通常需要较长的时间，尤其是在大数据量的情况下，可能需要结合大批量的离线处理和实时计算才可能满足需求。

数据挖掘的实际增效也是我们在进行大数据价值挖掘之前需要仔细评估的问题。并不见得所有的数据挖掘计划都能得到理想的结果。首先需要保障数据本身的真实性和全面性，如果所采集的信息本身噪声较大，或者一些关键性的数据没有被包含进来，那么所挖掘出来的价值规律也就大打折扣。其次也要考虑价值挖掘的成本和收益，如果对挖掘项目投入的人力物力、硬件及软件平台耗资巨大，项目周期也较长，而挖掘出来的信息对于企业生产决策、成本效益等方面的贡献不大，那么片面地相信和依赖数据挖掘的威力，也是不切实际和得不偿失的。

6. 存储及安全问题

在大数据的存储及安全保障方面，大数据由于存在格式多变、体量巨大的特点，也带

来了很多挑战。针对结构化数据，关系型数据库管理系统 RDBMS 经过几十年的发展，已经形成了一套完善的存储、访问、安全与备份控制体系。由于大数据的巨大体量，也对传统 RDBMS 造成了冲击，如前所述，集中式的数据存储和处理也在转向分布式并行处理。大数据更多的时候是非结构化数据，因此也衍生了许多分布式文件存储系统、分布式 NoSQL 数据库等来应对这类数据。然而这些新兴系统，在用户管理、数据访问权限、备份机制、安全控制等各方面还须进一步完善。安全问题，如果简而言之，一是要保障数据不丢失，对海量的结构、非结构化数据，需要有合理的备份冗余机制，在任何情况下数据不能丢失。二是要保障数据不被非法访问和窃取，只有对数据有访问权限的用户，才能看到数据，拿到数据。由于大量的非结构化数据可能需要不同的存储和访问机制，因此要形成对多源、多类型数据的统一安全访问控制机制，还是亟待解决的问题。大数据由于将更多、更敏感的数据汇集在一起，对潜在攻击者的吸引力更大，若攻击者成功实施一次攻击，将能得到更多的信息，“性价比”更高，这些都使得大数据更易成为被攻击的目标。LinkedIn 在 2012 年 650 万用户账户密码泄露；雅虎遭到网络攻击，致使 45 万用户 ID 泄露。2011 年 12 月，CSDN 的安全系统遭到黑客攻击，600 万用户的登录名、密码及邮箱遭到泄露。

7. 互连互通与数据共享问题

大数据要发挥威力，需要融合多行业的数据分析决策，这在智慧城市建设中尤其重要。为实现跨行业的数据整合，需要制定统一的数据标准、交换接口以及共享协议，这样不同行业、不同部门、不同格式的数据才能基于一个统一的基础进行访问、交换和共享。对于数据访问，还须制定细致的访问权限，规定什么样的用户在什么样的场景下，可以访问什么类型的数据。在大数据及云计算时代，不同行业、企业的数据可能存放在统一的平台和数据中心之上，需要对一些敏感信息进行保护，比如涉及企业商业机密及交易信息方面的数据，虽然是依托平台来进行处理，但是除了企业自身的授权人员之外，要保证平台管理员以及其他企业都不能访问此类数据。

1.4 大数据处理流程

整个大数据的处理流程可以定义为：在合适工具的辅助下，对广泛异构的数据源进行抽取和集成，结果按照一定的标准进行统一存储，并利用合适的数据分析技术对存储的数据进行分析，从中提取有益的知识并利用恰当的方式将结果展现给终端用户。具体来说，从数据源到数据的最终应用，其中的处理流程可以分为数据抽取与整合、数据分析、数据可视化。根据大数据处理的整个流程我们可以引出大数据所需的基础技术。大数据的处理流程如图 1.1 所示。

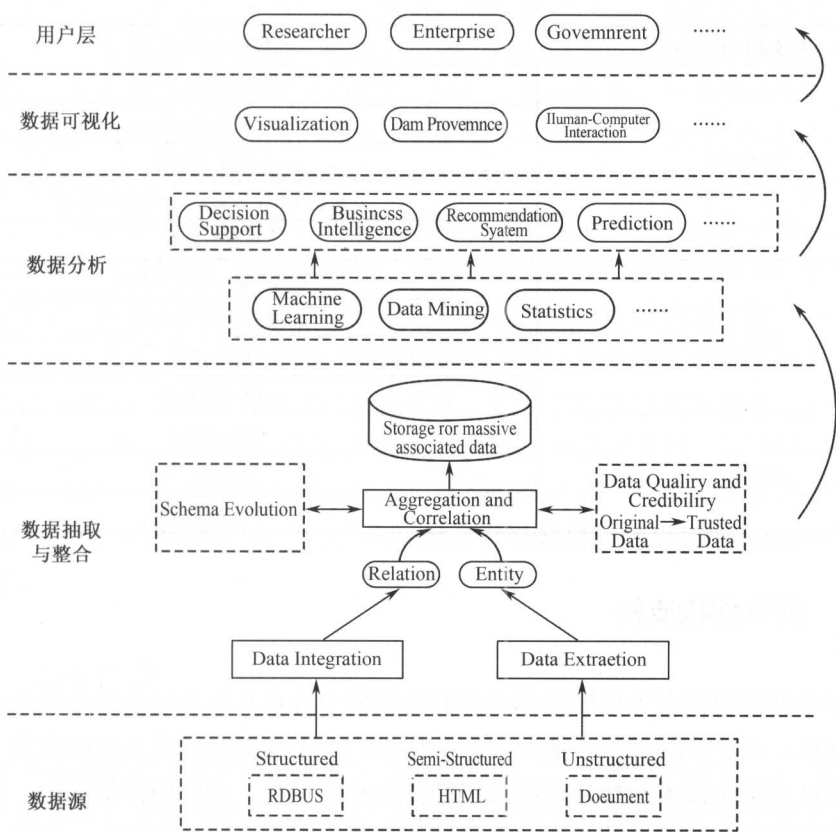


图 1.1 大数据的处理流程

1.5 大数据技术

我们依据以上的大数据处理流程，总结梳理出大数据的技术栈，见表 1.1。

表 1.1 大数据技术栈

大数据技术分类	大数据技术与工具
基础架构支持	云计算平台
	云存储
	虚拟化
	网络
	资源监控
数据采集	数据总线
	ETL 工具

续表

大数据技术分类	大数据技术与工具
数据存储	分布式文件系统
	关系型数据库
	NoSQL 数据库
	关系型数据库与非关系型数据库融合
	内存数据库
数据计算	数据查询、统计与分析
	数据预测与挖掘
	图谱处理
	BI 商业智能
展现与交互	图形与报表
	可视化工具
	增强现实技术

1.5.1 基础架构支持

大数据处理需要拥有大规模物理资源的云数据中心和具备高效的调度管理功能的云计算平台的支撑。云计算管理平台能为大型数据中心及企业提供灵活高效的部署、运行和管理环境，通过虚拟化技术支持异构的底层硬件及操作系统，为应用提供安全、高性能、高可扩展、高可靠和高伸缩性的云资源管理解决方案，降低应用系统开发、部署、运行和维护的成本，提高资源使用效率。

云计算平台可分为 3 类：以数据存储为主的存储型云平台，以数据处理为主的计算型云平台以及计算和数据存储处理兼顾的综合云计算平台。目前在国内外已经存在较多的云计算平台，开源的有 OpenStack、CloudStack、Apache Hadoop、10gen MongoDB、Abiquo AbiCloud、加利福尼亚大学的 Eucalyptus 项目、Enomalism 弹性云计算平台以及科学云计算平台 Nimbus 等。商业化的云计算平台有 Google 的 Google AppEngine，核心技术包括 MapReduce、Bigtable、GFS；微软的 Azure 平台；Amazon 有 EC2、S3、SimpleDB、SQS；Oracle EC2 上的 Oracle 数据库、Oracle VM、Sun xVM；Salesforce 的 Force.com 服务；EMC 的 Atoms 云存储系统；阿里云；中国移动的 BigCloud 大云平台等。

1.5.2 数据采集

足够的数据量是企业大数据战略建设的基础，因此数据采集是大数据价值挖掘中的重要的一环，其后的分析挖掘都建立在数据采集的基础上。

数据的采集有基于物联网传感器的采集，也有基于网络信息的数据采集。比如在智能交通中，数据的采集有基于 GPS 的定位信息采集、基于交通摄像头的视频采集、基于交通

卡口的图像采集、基于路口的线圈信号采集等。而在互联网上的数据采集是对各类网络媒介,如搜索引擎、新闻网站、论坛、微博、博客、电商网站等的各种页面信息和用户访问信息进行采集,采集的内容主要有文本信息、URL、访问日志、日期和图片等。之后我们需要把采集到的各类数据进行清洗、过滤、去重等各项预处理并分类归纳存储。

在分布式系统中,经常需要采集各个节点的日志,然后进行分析。在数据量呈爆炸式增长的今天,数据的种类丰富多样,也有越来越多的数据需要将存储和计算放到分布式平台。数据采集过程中的 ETL 工具将分布的、异构数据源中的不同种类和结构的数据抽取到临时中间层后进行清洗、转换、分类、集成,最后加载到对应的数据存储系统,如数据仓库或数据集中,成为联机分析处理、数据挖掘的基础。企业每天都会产生大量的日志数据,对这些日志数据的处理需要特定的日志系统。因为与传统的数据相比,大数据的体量巨大,产生速度非常快,对数据的预处理需要实时快速,因此在 ETL 的架构和工具选择上,也需要采用分布式内存数据、实时流处理系统等现代信息技术。根据实际生活环境中应用环境和需求的不同,目前已经产生了一些高效的数据采集工具,包括 Flume、Scribe、Chukwa 和 Kafka 等。

1.5.3 数据存储

云计算中的数据存储是实现云计算系统架构中的一个重要组成部分。云存储专注于解决云计算中海量数据的存储问题,它既可以给云计算技术提供专业的存储解决方案,又可以独立发布存储服务。云存储将存储作为服务,它将分别位于网络中不同位置的大量类型各异的存储设备通过集群应用、网格技术和分布式文件系统集合起来协同工作,通过应用软件进行业务管理,并通过统一的应用接口对外提供数据存储和业务访问功能。目前,云存储的兴起正在颠覆传统的存储系统架构,其正以良好的可扩展性、性价比和容错性等优势得到业界的广泛认同。云存储系统具有良好的可扩展性、容错性,以及内部实现对用户透明等特性,这一切都离不开分布式文件系统的支撑。现有的云存储分布式文件系统包括 Google GFS、Hadoop HDFS、Lustre、FastDFS、Clemson 大学的 PVFS、Sun PFS、加州大学 Santa Cruz 分校 Sage Weil 设计的 Ceph 和 Taobao TFS 等。

目前存在的数据库存储方案有 SQL、NoSQL 和 NewSQL。

SQL 是目前为止企业应用中最成功的数据存储方案,仍有相当大一部分的企业把 SQL 数据库作为数据存储方案。关系型数据库能够较好地保证事务的 ACID 特性,但在可扩展性、可用性等方面,表现出较大的不足,并且只能处理结构化的数据,面对数据的多样性、处理数据的实时性等方面,都不能满足大数据时代环境下数据处理的需要。使用较多的 SQL 产品有 IBMDB2、ORACLE、MySQL、MSSQL Server 等。

NoSQL 是为了解决 SQL 的不足而产生的。但它在设计时放松了事务的 ACID 特性。根据 CAP 定理, NoSQL 数据库不可能同时满足一致性(Consistency)、可用性(Availability)和分区容错性(Partition tolerance)三个特性。NoSQL 数据库在设计时经常会保证分区容

错性，而牺牲一致性和可用性，因而 NoSQL 的应用范围也受到了很大的限制。如何构建具有高可扩展性、高可用性、高性能的，同时还能保证 ACID 事务特性的数据库就成为了新的发展方向。现有的 NoSQL 数据库有很多，例如 HBase、Cassandra、MongoDB、CouchDB、Hypertable、Redis 等。

NewSQL 是为解决上述数据库存在的不足，顺应科技发展的产物。该类数据库要求，不仅要具有 NoSQL 对海量数据的存储管理能力，还要保持对传统数据库支持 ACID 和 SQL 等特性。目前 NewSQL 系统产品有 H-Store、VoltDB、NuoDB、TokudB、MemSQL 等。

1.5.4 数据计算

在大数据的环境下，数据计算除了标准的查询、统计、分析之外，主要还体现在数据挖掘、深度学习、社交计算、计算广告等几个方面。

数据挖掘又称从数据库中发现知识（KDD）、数据分析、数据融合以及决策支持。数据挖掘领域已经有了较长时间的发展，但随着研究的不断深入、应用的愈发广泛，数据挖掘的关注焦点也逐渐有了新的变化。其总的趋势是数据挖掘研究和应用更加“大数据化”和“社会化”。在用户层面，移动计算设备的普及与大数据革命带来的机遇使得搜索引擎对用户所处的上下文环境具有了前所未有的深刻认识，但对于如何将认识上的深入转化为用户信息获取过程的便利仍然缺乏成功经验。近年来，以用户个性化、用户交互等为代表的研究论文的数量大幅增加。除此之外，社交网络服务的兴起对互联网数据环境和用户群体均将形成关键性的影响，如何更好地面对相对封闭的社交网络数据环境和被社交关系组织起来的用户群体，也是数据挖掘面临的机遇与挑战。

深度学习是机器学习研究中的一个新的领域。它在于建立模拟人脑进行分析学习的神经网络，模仿人脑机制来解释一些特定类别的数据，例如图像、语音和文本。它是无监督学习的一种。深度学习的主要思想是增加神经网络中隐藏层的数量，使用大量的隐藏层来增强神经网络对特征筛选的能力，以增加网络层数的方式来取代之前依赖人工技巧的参数调优，从而能够用较少的参数表达出复杂的模型函数，从而逼近机器学习的终极目标——知识的自动发现。

社交网络每天都会产生大量的用户数据，它吸引着无数研究者从无序的数据中发掘有价值的信息。在社交网络的分析与研究过程中，会利用到社会学、心理学甚至是医学的基本理论来作为指导。社交网络上的传播模型，虚假信息和机器人账号的识别，基于社交网络信息对股市、大选以及传染病的预测，社区圈子的区别，社交网络中人物的影响力等，都可以作为社交网络中的研究课题。通过人工智能领域的机器学习、图论等算法对社交网络中行为和未来的趋势进行模拟和预测。

计算广告是一门正在兴起的分支学科。它由信息科学、统计学、计算机科学以及微观经济学等学科交叉融合而成。它涉及大规模搜索和文本分析、信息获取、统计模型、机器学习、分类、优化及微观经济学。计算广告学所面临的最主要挑战是在特定语境下特定用户和相应的广告之间找到“最佳匹配”。语境可以是用户在搜索引擎中输入的查询词，也

可以是用户正在读的网页，还可以是用户正在看的电影等。而用户相关的信息可能非常多也可能非常少。潜在广告的数量可能达到几十亿。因此，取决于对“最佳匹配”的定义，面临的挑战可能导致在复杂约束条件下的大规模优化和搜索问题。

面向大数据处理的数据查询、统计、分析、挖掘等计算需求，促生了大数据计算的不同计算模式，整体上我们把大数据计算分为离线批处理计算和实时计算两种。

离线批处理计算模式最典型的应该是 Google 在 2004 年提出的 MapReduce 编程模型。

MapReduce 的核心思想如下。

① 将大数据并行处理问题分而治之，即将一个大数据通过一定的数据划分方法，分成多个较小的具有同样计算过程的数据块，数据块之间不存在依赖关系，将每一个数据块分给不同的节点去处理，最后再将处理的结果进行汇总。

② 上升到抽象语言模型 Map 和 Reduce。将对大量顺序式数据元素或者记录进行扫描和对每个数据元素或记录做相应的处理并获得中间结果信息的两个过程抽象为 Map 操作；将对中间结果进行收集整理和产生最终结果并输出的过程抽象为 Reduce 操作。

③ 以统一架构为程序员隐藏系统层细节，MapReduce 提供的统一框架实现了自动并行化计算，可负责自动完成多种系统底层的相关处理，如计算任务的自动划分和调度，数据的自动化分布存储和划分，处理数据与计算任务的同步，结果数据的收集整理，系统通信、负载平等、计算性能优化处理，处理系统节点出错检测和失效恢复等，这些自动实现的并行计算，为程序员隐藏了系统层细节。

实时计算最重要的一个需求是能够实时响应计算结果，一般要求为秒级。主要有以下两种应用场景：一种是数据源是实时的、不间断的，同时要求对用户请求的响应时间也是实时的；另一种是数据量大，无法进行预算，但要求对用户请求实时响应。

实时计算的过程一般可以分为 3 个阶段：数据的产生与采集、数据的实时计算、实时查询服务。

数据的实时采集阶段，要保证可以完全地采集到所有的日志数据，为实时应用提供实时数据。响应的时间也要保证实时性，1 秒左右的低延迟，系统的配置简单、部署容易、可靠稳定。目前，已经有较多的互联网企业有自己的数据采集工具，主要有 Facebook 的 Scribe、Cloudera 的 Flume、LinkedIn 的 Kafka、淘宝的 TimeTunnel、Hadoop 的 Chukwa 等，他们都可以满足每秒对数百兆日志数据的采集和传输需求。

数据的实时计算，包括数据的传输与分析计算。在流数据不断变化的运动过程中实时地进行分析，捕捉到可能对用户有用的信息，并把结果发送出去。整个过程中，数据分析处理系统是主动的，而用户却处于被动接收的状态。数据的实时计算框架需要能够适应流式数据的处理，可以进行不间断的查询，同时要求系统稳定可靠，具有较强的可扩展性和可维护性。目前较为主流的实时流计算框架包括 Yahoo 的 S4、Twitter 开源的 Storm，还有 Esper、Streambase、HStreaming 等。

实时查询服务，是存储对象对外提供服务的过程。包括 3 种类型的数据库：一是全内存查询，其直接提供数据读取服务，定期转存到磁盘或者数据库，进行持久化；二是半内存查询，主要有 Redis、Memcache、MongoDB、BerkeleyDB 等内存数据库提供数据实时查

询服务，由这些系统进行持久化操作；三是全磁盘查询，使用 HBase 等以分布式文件系统（HDFS）为基础的 NoSQL 数据库，对于 key-value 引擎，关键是设计好 key 的分布。

1.5.5 展现与交互

计算结果需要以简单直观的方式展现出来，才能最终为用户所理解和使用，形成有效的统计、分析、预测及决策，应用到生产实践和企业运营中，因此大数据的展现技术，以及数据的交互技术在大数据全局中也占据重要的位置。

Excel 形式的表格和图形化展示方式是人们熟知和使用已久的数据展示方式，也为日常的简单数据应用提供了极大的方便。华尔街的很多交易员还都依赖 Excel 和他们很多年积累和总结出来的公式来进行大宗的股票交易，而微软公司和一些创业者也看到市场潜力，在开发以 Excel 为展示和交互方式、结合 Hadoop 等技术的大数据处理平台。

人脑对图形的理解和处理速度大大高于文字。因此，通过视觉化呈现数据，可以深入展现数据中的潜在的或复杂的模式和关系。随着大数据的兴起，也涌现了很多新型的数据展现和交互方式，和专注于这方面的一些创业公司。这些新型方式包括交互式图表，可以在网页上呈现，并支持交互，可以操作、控制图标，进行动画演示。另外交互式地图应用如 Google 地图，可以动态标记、生成路线、叠加全景航拍图等，由于其开放的 API 接口，可以与很多用户地图和基于位置的服务应用结合，因而获得了广泛的应用。Google Chart Tools 也给网站数据可视化提供了很多种灵活的方式。从简单的线图、Geo 图、Gauges（测量仪），到复杂的树图，Google Chart Tools 提供了大量设计优良的图表工具。

大数据时代也诞生了很多新兴的大数据可视化技术及相应的创业公司，能够将数据所蕴含的信息与可视化展示有机地结合起来的“信息图”方式，目前大行其道。诞生于斯坦福大学的大数据创业公司 Tableau 能够将数据运算与美观的图表完美地结合在一起。Tableau 的设计与实现理念是：界面上的数据越容易操控，公司对自己所在业务领域里的所作所为到底是正确还是错误，就能了解得越透彻。快速处理，便捷共享，是 Tableau 的另一大特性，这将使得用户使用数据的积极性大大增加。另一家大数据可视化创业公司 Visually 以丰富的信息图资源而著称，它是一个社会化的信息图创作分享平台。很多用户乐意把自己制作的信息图上传到网站中与他人分享，信息图极大地刺激视觉表现，促进用户间相互学习、讨论。

此外，3D 数字化渲染技术也被广泛地应用在很多领域，如数字城市、数字园区、模拟与仿真、设计制造等，具备很高的直观操作性。现代的增强现实 AR 技术通过电脑技术，将虚拟的信息应用到真实世界，真实的环境和虚拟的物体实时地叠加到同一个画面或空间同时存在。结合虚拟 3D 的数字模型和真实生活中的场景，提供了更好的现场感和互动性。通过 AR 技术，用户可以和虚拟的物体进行交互，如试戴虚拟眼镜、试穿虚拟衣服、驾驶模拟飞行器等。在德国，工程技术人员在进行机械安装、维修、调式时，通过头盔显示器，可以将原来不能呈现的机器内部结构，以及它的相关信息、数据完全呈现出来。

现代的体感技术，如微软的 Kinect 以及 Leap 公司的 Leap Motion 体感控制器，能够检

测和感知到人体的动作及手势，进而将动作转化为对电脑及系统的控制，使人们摆脱了键盘、鼠标、遥控器等传统交互设备的束缚，直接用身体和手势来与电脑和数据交互。当今热门的可穿戴式技术，如 Google 眼镜，则有机地结合了大数据技术、增强现实及体感技术。随着数据的完善和技术的成熟，我们可以实时地感知我们周围的现实环境，并且通过大数据搜索、计算，实现对周围的建筑、商家、人群、物体的实时识别和数据获取，并叠加投射在我们的视网膜上，这样可以实时地帮助我们工作、购物、休闲等，提供极大的便利。当然这种新型设备和技术的弊端也显而易见，我们处在一个随时被监控，隐私被刺探、侵犯的状态，所以大数据技术所带来的安全和隐私性问题也不容忽视。

1.6 练习题

1. 大数据有哪几个特征？
2. 大数据存在哪几个方面的问题？
3. 大数据技术分为哪几大类？
4. 大数据处理流程会用到哪些更新的技术？

参考文献

- [1] 陶翔，罗天雨. 大数据技术的发展历程及演化趋势. 科技日报, 2014. 08. 10.
- [2] 大数据的前世今生——大数据的特征与发展历程. <http://www.leiphone.com/news/201410/NgTsZw3yDjEbk9on.html>.
- [3] 大数据的 13 个应用场景. [http://www.datatang.com/news/details_1331.htm?from= timeline &isappinstalled =0](http://www.datatang.com/news/details_1331.htm?from=timeline&isappinstalled=0).
- [4] 36 大数据. <http://www.36dsj.com/>.
- [5] 大数据应用. <http://www.36dsj.com/archives/category/application-area>.
- [6] 大数据在金融行业的应用. <http://www.36dsj.com/archives/16816>.
- [7] 大数据在医疗行业的 15 个应用场景. <http://www.36dsj.com/archives/13691>.
- [8] 埃博拉与大数据. <http://www.36dsj.com/archives/15395>.
- [9] 从银行、保险到证券，揭开大数据在金融行业的应用. <http://www.36dsj.com/archives/16023>.
- [10] 大数据时代的高考择校. <http://www.36dsj.com/archives/16482>.
- [11] 大数据在教育领域如何应用. <http://www.36dsj.com/archives/4002>.
- [12] 林子雨. 大数据技术基础. 厦门大学计算机科学系, 2013. 9.
- [13] 赵勇，林辉，沈寓实等. 大数据革命—理论、模式与技术创新. 北京：电子工业出版社, 2014.

第2章

大数据基础支撑——数据中心及云计算

大数据技术正在改变目前计算机的运行模式，正在改变着这个世界。它能处理几乎各种类型的海量数据，无论是微博、文章、电子邮件、文档、音频、视频，还是其他形态的数据。它工作的速度非常快，可以达到实时。而为大数据提供核心基础支撑的是数据中心的大规模计算、存储及网络资源，以及负责管理、调度、监控这些资源的云计算平台。云计算让用户能够按照他们的业务需求获取相应的计算力、存储空间和信息服务，将计算任务分布到大量服务器构成的资源池上。云计算及其技术给了人们廉价获取巨量计算和存储的能力，云计算分布式架构则能够很好地支持大数据存储和处理需求。这样的低成本硬件+低成本软件+低成本运维，更加经济和实用，为大数据处理和利用提供强力支撑。

在本章中，我们将重点介绍数据中心和云计算的概念、大数据和云计算的关系、云资源调度与管理、开放云计算平台 OpenStack 等内容。

2.1 数据中心概述

数据中心起源于 20 世纪 60 年代，发展至今，数据中心先后经历了计算中心、信息中心和服务中心 3 个发展阶段。维基百科给出的定义是“数据中心是一整套复杂的设施，它不仅仅包括计算机、系统和其他与之配套的设备（例如通信和存储系统），还包含冗余的数据通信连接、环境控制设备、监控设备以及各种安全装置”。目前数据中心在各行业都发挥着至关重要的作用，承载着企业的关键业务，为用户提供及时可靠的数据存储、数据检索、数据分析及发掘、高性能计算等服务，如 Google 数据中心为全球网民提供搜索、视频等服

务。随着云计算的发展,IT资源的应用和共享方式发生了巨大的变化。云计算是网格计算、并行计算、分布式计算、虚拟化、负载均衡等传统计算机和网络技术发展融合的产物。它是一种全新的计算方式和资源使用方式,普通用户可以十分方便地接入强大的IT资源并按需部署自己的服务,同时多种全新的业务模式能够得以实现,另外IT资源和服务能够从底层基础设施中抽象出来,这极大增强了资源的共享性和灵活性。

从数据中心模式服务的发展而言,其产生和演化经历了三个阶段:包括主机共享时期,主机托管时期,应用服务托管时期。起初就是主机托管服务,只为用户提供电源、带宽,机器重新启动都要自己来做;随后出现了主机托管服务,主要是带宽上有保证,电源上有备份,并且可以部分代为管理;一些大型的客户要求更多的增值服务,包括一些关键性业务,如要求安全性、数据流的分析、资源的占用状况等,需求越来越多,要求有更多的服务。在这种情况下,出现了提供综合服务的大型数据中心服务商。这个时期比较成熟的数据中心模式才算正式出现。

数据中心是云计算的实现平台,云计算时代的数据中心已经从原本的数据存储节点转变为面向服务和应用的IT核心节点。随着各种数据密集型业务的出现,数据中心已经成为唯一能够支持大规模云计算应用的服务平台(例如Microsoft Azure、Amazon EC2、Google Search、Facebook等)。同时,为了给云计算提供“无限可能”的资源池,数据中心必须包含更多存储资源、计算资源以及通信带宽。新一代数据中心将包含数万数十万台服务器,例如,目前Google在全球有30多个大型数据中心,单个数据中心服务器数目超过了45 000台,微软在印第安纳州建立的数据中心投资规模达6.7亿美元,在计划构建的数据中心可容纳的服务器数目高达300 000台,国内的大型互联网公司如阿里巴巴、腾讯新建的数据中心规模也都超过200 000台服务器。

以下将从几方面对数据中心在云计算时代面临的问题、关键技术以及业界的发展动态逐一分析。

2.1.1 云计算时代数据中心面临的问题

随着云时代的到来,云数据中心也变得炙手可热,相比之下,传统数据中心将面临被淘汰的形势,目前传统的数据中心主要面临以下几个问题。

网络架构问题:传统数据中心网络采用三层结构,所需网络设备多,平均时延长。另外,随着存储网络和数据网络的融合,存储流量对时延要求更为严格,三层结构带来的高时延问题需要通过扁平化网络结构来解决。

融合性问题:数据网络与存储网络的分离现状阻碍数据中心的发展,如何实现网络的有效融合,对于数据中心发展至关重要。

云计算下业务高带宽需求:数据中心将处理视频、数据发掘、高性能计算等高带宽业务,突发流量现象较多,因而要求网络必须保证数据能够高速率传输。

虚拟化问题:为了解决当前数据中心设备利用率低的问题,需要采用各种虚拟化技术,

从而提高设备利用率。

高可用性：随着数据中心规模的扩大，如何保证在链路、设备或网络故障及人为操作失误时能够实现服务不中断成为日益关注的一个问题；网络扩展或升级时，网络能够正常运行，对网络性能影响不大。

安全性问题：数据中心的业务具有高开放性、多业务并存以及不确定的访问来源等特点，因此数据中心往往面临着较多的安全威胁。当前云计算没有成熟的安全防护技术，如何提高数据中心安全性是一个迫切要解决的问题。

低能耗问题：构建及运营数据中心所需的能耗过大，尽快形成绿色节能、高效运行的数据中心势在必行。

因此，构建新一代数据中心刻不容缓，新一代数据中心应能够很好地解决上述问题，为用户提供高带宽、低时延、低能耗、高效率、安全可靠的服务，保证企业单位及服务提供商构建数据中心的成本低、设备利用率高、数据中心可用性高，实现高效管理。根据上述需求，基于云计算的新一代数据中心应运而生，将云计算与数据中心有效结合，实现优势互补，充分为现代企业应用服务。

2.1.2 新一代数据中心关键技术

数据中心在现代生产及生活中持续发挥越来越重要的作用，当前数据中心能够满足云计算提出的新的应用需求，提高服务能力，保证服务的高可靠性、高效率，降低企业单位构建和运营数据中心的成本，实现绿色节能的目标。以下对影响数据中心部署和运营的关键技术逐一分析，主要包括网络架构设计、网络融合技术、节能技术、虚拟化技术及安全策略等。

1. 网络架构设计

传统网络架构基于的树形结构是基于二叉树构建的，采用垂直扩展方式，通过添加更高的层数及更高性能的交换机设备实现扩展。但该拓扑难以克服传统树形结构的固有缺陷：流量在核心根节点处汇集，出现热点，核心节点容易成为网络性能的瓶颈；网络存在严重过载问题，高层采用高性能、高容量的交换机只能在一定程度上缓解过载，不能从根本上解决过载和热点问题；网络的扩展能力很受限，通过采购高端口密度的高性能设备，导致设备开销巨大，不利于构建大规模的数据中心。因此，基于未来数据中心对拓扑的高带宽、低时延、高可靠性、运营开销低、管理方便等要求，涌现出不少新的拓扑结构，如 Fat-Tree、VL2、BCube、DCell 等。Fat-Tree 通过在核心交换机处添加“粗”链路，解决热点问题，但扩展能力局限于核心交换机的端口数目；VL2 在解决热点问题及扩展能力上具有很好的性能，但仍具有布线复杂的缺点；DCell 结构具有很好的扩展性能，能够满足数据中心随着业务拓展及应用动态发展的需求，但是流量分布不均匀，同时扩展粒度过大；BCube 结构是基于数据中心集装箱思想而设计的网络拓扑结构，适应于目前主流的 Data Center in a Box

的构建思想,网络性能良好,但也存在扩展粒度过大的问题。随着数据中心研究的不断深入,业务的不断扩展,现有拓扑在提供高带宽、低时延的网络性能及部署虚拟机方面已经显示出一定的不足,难以保证未来数据中心的发展对于网络设计高带宽、低时延、高可靠性、高灵活性、低能耗的要求。

2. 网络融合技术

当前传统数据中心发展模式已经严重阻碍数据中心的发展,以太网、存储网络及高性能计算网络融合是数据中心网络发展趋势,通过融合可以降低成本、降低管理复杂度、提高安全性等。现阶段支持三网融合的关键技术主要有 FCoE(光纤以太网通道技术)、DCB(数据中心桥接技术)及 TRILL(多链接透明互连)等。FCoE(Fiber Channel Over Ethernet)以太网光纤通道,通过在以太网上传输 FC 的数据,从而实现 I/O 接口整合,减少数据中心的复杂性。DCB(DataCenter Bridging,数据中心桥接)是数据中心内部“三网融合”的关键技术,也被称为 CEE(Convergence Enhanced Ethernet,融合增强型以太网)。其核心是将以以太网发展成为拥有阻塞管理和流量控制功能的低延迟和不丢弃数据包的传输技术,从而拥有以太网的低成本、可扩展和 FC 的可靠性。TRILL(多链接透明互连)由 IETF 推进,属于二层标准,其核心是为克服生成树协议(STP)在规模上和拓扑重聚方面存在的不足。TRILL 是一个基于最短路径架构路由的多跳标准以太网网络协议,主要是整合了网桥和路由器的优点,将链路状态路由技术应用在二层,提高对单播和多播在多路径方面(Multi-Pathing)的支持,并降低延迟。

3. 节能技术

当前数据中心运行能耗及制冷能耗开销甚大,给企业增加过大的负担,也不利于资源节约及环境保护,因此新一代的数据中心的一项要求是降低能耗。在建设数据中心的地点选择上各大公司多数考虑环境温度较低的地点,从而利用当地适宜的气候用新鲜空气进行冷却。供电方面,对新一代数据中心的供电可以采用风能、太阳能等清洁能源,减少碳排放,应对全球气候变暖问题。例如,谷歌在北达科他州的数据中心使用 NextEra Energy Resources 的风力发电提供能源。思科推出了 EnergyWise 技术能够帮助企业获得和判断设备能耗和运行状况,通过可控性网络管理降低处于静态或动态的装置能耗,从而降低网络管理的复杂性。华三 S12500 通过智能化的 EMS 引擎系统,支持对电源的智能管理功能,降低系统能耗。

4. 虚拟化技术

虚拟化技术是将硬件资源抽象化,经整合之后再分配,以达到资源设备的便捷、高效的使用。按虚拟化技术的应用特点,虚拟化技术主要分为以下几类:服务器虚拟化、存储虚拟化、网络虚拟化及桌面虚拟化。将虚拟化技术应用于数据中心领域,能够解决阻碍数据中心发展的诸多问题,提高物理设备的利用率,有效降低数据中心运维成本,降低能耗以及保证数据中心服务的可靠性、连续性。但是在数据中心中应用虚拟化技术也存在一些

问题，比如当前业界没有统一的虚拟化标准平台和开放协议，移植和管理工具尚不成熟，以及虚拟化运作存在一定风险等。

5. 安全策略

作为集中了用户最重要信息的资产，数据中心安全的重要性不言而喻，常见的攻击类型包括应用层攻击、网络层攻击以及对网络基础设施的攻击等。数据中心对网络边界层面的防护包括对非法访问的阻断、对访问用户身份的识别、监测攻击情况以及防止数据被篡改等。

2.1.3 业界发展动态

根据赛迪的报告，中国数据中心市场每年按照 50% 的增长率在增长，2006 年国内市场 23 亿元，2007 年国内市场 36 亿元，2008 年国内市场 53 亿元，2009 年国内市场 76 亿元，2010 年国内市场 100 亿元，按照这个发展趋势，中国到 2020 年的数据中心市场规模将达到 5766 亿元人民币，再次佐证这个市场具有非常大的潜力。随着数据中心应用领域不断扩展，应用不断深入，无论是网络设备供应商、服务提供商，还是各大门户网站纷纷将关注点集中在数据中心的研究和开发领域，期望在数据中心在新技术应用领域具有一席之地，为客户提供构建数据中心的设备、方案设计及服务。

在电信领域，数据中心已经成为保证服务运营商提供高速率、可靠服务的核心。根据现阶段 3G 网络的广泛部署及未来 4G 网络的发展需求，致使目前的数据中心承载的业务变得更大，难以保证用户对服务质量的要求，传统数据中心难以实现下一代数据中心的发展目标，因此构建新一代数据中心是各运营商当前的重要任务之一。基于应用及市场发展需求，不少服务运营商均推出构建新一代数据中心的计划。中国移动在过去 3 年内（2010—2012 年）在多个省份建设数据中心用于支撑企业信息化业务和移动互联网业务；联通提出新一代数据中心的规划，不少省分公司已经开始尝试利用虚拟化技术实现部分增值业务，并取得了初步的成效；中国电信广东分公司在广东建设 2 万平方米的下一代绿色节能云计算数据中心，为用户提供先进的云计算及数据中心服务。上海电信选择与微软、惠普合作，建设新一代的“云数据中心”，利用虚拟化的技术，建立 IT 基础架构。各国的电信公司这两年也在针对云计算的需求提出新的数据中心建设要求。AT&T 公司将投资 1.2 亿美元部署伊利诺伊州和美国东海岸新一代数据中心，该数据中心针对公司的无线业务，包括 3G 和 4G 服务；德国电信将在马格德堡建立的云计算数据中心将成为德国最大的高安全性数据中心之一，使用面积约为 24000 平方米。

为了支持云计算时代的数据中心建设，各大设备制造商也相继推出了针对云计算时代的数据中心设计方案和产品系列。思科作为世界上最大的网络设备提供商，提出 Data Center 3.0 构想，旨在实现融合、自动化、高带宽、高效管理、绿色节能的目标，推出适应于云计算时代数据中心互连的交换机系列，包括 Cisco Nexus 7000、5000、3000、2000、1000 系

列。H3C 推出 H3C S12500 系列核心路由交换机及 H3C S10500 系列核心路由交换机,采用 CLOS 多级多平面交换架构,实现大容量交换,实现网络基础架构的统一、安全策略的统一部署及数据中心资源的统一管理,为用户提供高效、安全的服务。中兴通讯用于数据中心的交换机主要有 ZXR10 8900E 系列核心交换机、ZXR10 M6000 交换机、ZXR10 5900E 交换机和 ZXR10 T128/T64E 交换机。交换机系列采用 Crossbar 空分交换结构,主要用于数据中心的中心和汇集层互连。

Facebook 2011 年在其总部举行发布会,宣布启动名为开放计算项目(The Open Compute Project)的开源方案(<http://opencompute.org/servers/>)(<http://opencompute.org/datacenters/>),将其底层服务器和数据中心技术全部开源。

Facebook 不仅开放了他们领先业界的定制服务器和数据中心技术的规范和文档,甚至连服务器和数据中心的 CAD 图纸设计也完全公开。以此回馈社区,促进生态系统成长,推进技术进步。Facebook 的数据表明,他们的定制硬件各方面指标比业界标准要高得多,与公司从设备厂商购买的同类现成产品相比,效率提升 38%,成本则降低了 28%。而整个数据中心的能耗按 PUE (Power Usage Effectiveness, 电能使用效率)衡量是 1.07,大大低于业界通常的 1.5。

Facebook 此举堪与 2004 年 Google 陆续发表 MapReduce 等论文相提并论。Google 当年的开放措施直接启动了如今如火如荼的云计算运动。而 Facebook 则将云计算之火烧到了服务器硬件和数据中心领域。

2.1.4 小结

数据中心在现代信息化社会中发挥越来越重要的作用,当前必须保证数据中心能够满足云计算时代提出的新的应用需求,提高服务能力,保证服务的高可靠性、高效率,降低企业单位构建和运营数据中心的成本,实现绿色节能的目标。本节对云计算时代数据中心的新要求进行了总结,分析了未来发展的各项关键技术,包括网络结构设计、流控问题、虚拟化技术、交换架构的设计、网络融合技术、安全技术、绿色节能及高效的管理技术等。最后对业界的发展情况进行了总结。总之,新一代的数据中心将向着降低 IT 运营成本、提高管理效率、提高业务灵活性、提高安全性、提高可靠性以及降低能耗的目标发展。

2.2 云计算简介

随着信息和通信技术的快速发展,如图 2.1 所示,计算模式经历了从最初把任务集中交付给大型处理机模式,到后来发展为基于网络的分布式任务处理模式,再到最新的按需处理的云计算模式。最初的单个处理机模式处理能力有限,并且请求需要等待,效率低下。后来,随着网络技术的不断发展,按照高负载配置的服务器集群,在遇到低负载的时候,

会有资源的浪费和闲置，导致用户的运行维护成本提高。而云计算把网络上的服务资源虚拟化，整个服务资源的调度、管理、维护等工作由专门的人员负责，用户不必关心“云”内部的实现，因此云计算实质上是给用户像传统的电力、水、煤气一样的按需计算服务，它是一种新的有效的计算使用范式。并且，云计算是分布式计算、效用计算、虚拟化技术、Web 服务、网格计算等技术的融合和发展，其目标是用户通过网络能够在任何时间、任何地点最大限度地使用虚拟资源池，处理大规模计算问题。目前，在学术界和工业界共同推动之下，云计算及其应用呈现迅速增长的趋势，各大云计算厂商如 Amazon、IBM、Google、Microsoft、Sun 等公司都推出自己研发的云计算服务平台。而学术界也源于云计算的现实背景纷纷对模型、应用、成本、仿真、性能优化、测试等诸多问题进行了深入研究，提出了各自的理论方法和技术成果，极大地推动了云计算继续向前发展。

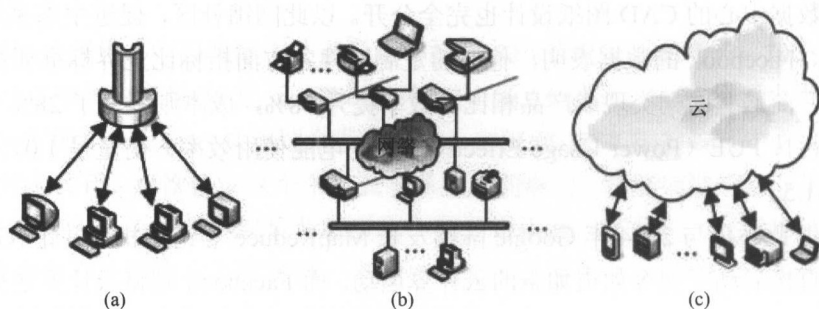


图 2.1 云计算模式的演化

2.2.1 云计算定义

云计算概念最早是由 Google 提出的，一方面是因为当时在网络拓扑图中用云来代表远程的大型网络，另一方面也用来指代通过网络应用模式来获取服务。狭义云计算是指 IT 基础设施的交付和使用模式，指通过网络以按需、易扩展的方式获得所需的资源；广义云计算是指服务的交付和使用模式，指通过网络以按需、易扩展的方式获得所需的服务。这种服务可以是 IT 和软件、互联网相关的，也可以是任意其他的服务，它具有超大规模、虚拟化、可靠安全等独特功效。

目前，不同文献和资料对云计算的定义有不同的表述，主要有以下几种代表性的定义。

定义 1：云计算是一种能够在短时间内迅速按需提供资源的服务，可以避免资源过度 and 过低使用。

定义 2：云计算是一种并行的、分布式的系统，由虚拟化的计算资源构成，能够根据服务提供者和用户事先商定好的服务等级协议动态地提供服务。

定义 3：云计算是一种可以调用的虚拟化的资源池，这些资源池可以根据负载动态重新配置，以达到最优化使用的目的。用户和服务提供商事先约定服务等级协议，用户以按时付费模式使用服务。

定义 4: 云计算是一种大规模分布式的计算模式, 由规模经济所驱动, 能够把抽象化的、虚拟化的、动态可扩展的计算、存储、平台及服务以资源池的方式管理, 并通过互联网按需提供给用户。

定义 1 强调了按需使用方式, 定义 2 中突出了用户和服务提供商双方事先商定的服务等级协议。这两个定义都从一定的角度给出定义。定义 3 和定义 4 综合了前面两种定义的描述, 更好地揭示了云计算的特点和本质。

2.2.2 云计算主要特征

云计算是一种按使用量付费的模式, 这种模式提供可用的、便捷的、按需的网络访问, 进入可配置的计算资源共享池(资源包括网络、服务器、存储、应用软件、服务), 这些资源能够被快速提供, 只需要投入很少的管理工作, 或服务供应商进行很少的交互。云计算有以下五个主要特征。

1. 按需自助服务

消费者可以单方面按需部署处理能力, 如服务器时间和网络存储, 而不需要与每个服务供应商进行人工交互。

2. 通过网络访问

可以通过互联网获取各种能力, 并可以通过标准方式访问, 以通过众多瘦客户端或富客户端推广使用(例如移动电话、笔记本电脑、PDA 等)。

3. 与地点无关的资源池

供应商的计算资源被集中, 以便以多用户租用模式服务所有客户, 同时不同的物理和虚拟资源可根据客户需求动态分配和重新分配。客户一般无法控制或知道资源的确切位置。这些资源包括存储、处理器、内存、网络带宽和虚拟机器。

4. 快速伸缩性

可以迅速、弹性地提供资源, 能快速扩展, 也可快速释放以实现快速缩小。对客户来说, 可以租用的资源看起来似乎是无限的, 并且可在任何时间购买任何数量的资源。

5. 按使用付费

能力的收费是基于计量的一次一付, 或基于广告的收费模式, 以促进资源的优化利用。比如计量存储, 带宽和计算资源的消耗, 按月根据用户实际使用收费。在一个组织内的云可以在部门之间计算费用, 但不一定使用真实货币。

云计算新的范式的特点带来了众多的优势, 同时引入了一些新的问题亟待解决。这些因素制约着云计算技术及其应用的发展, 见表 2.1。

表 2.1 云计算的优势和对应问题

云计算	优势	问题
安全性	缩短单机密集数据处理任务时间，把处理任务分配到各个节点计算，提高了效率	用户关注传输到云计算端的敏感处理数据是否安全
可靠性	减少用户购买物理硬件设备的费用，资源以服务的方式进行租赁，降低用户资金投入的前期风险，促进用户把精力投入业务中	虽然用户不需要维护软件、硬件，但是用户使用云计算服务的质量依赖云计算本身的质量
可维护性	提供专业的软件管理和维护服务，减少了普通用户软件平台的日常维护管理成本	是否所有的软件应用都适合在云计算环境下开发应用，而以往的软件应用如何移植到云计算环境下
交互性	用户可以根据业务需要动态地按需请求云计算服务，处理高峰期负载并在非高峰期释放资源	云计算服务提供商的实际扩展能力有限，需要多个云计算服务商间的交互，而云计算服务之间的交互性较差

2.2.3 Web 服务、网格和云计算

Web 服务、网格和云计算的很多地方有相似之处，并且云计算是前两者的演化、发展，因此各个概念间容易混淆。区分相关概念间的差异性，有助于理解和把握云计算的本质，见表 2.2，本节比较每个概念之间的特征，分析彼此间的相互关联。

1. 异构性

Web 服务仅支持软件层次上异构的服务，用户调用的服务可以是各种语言开发的功能模块，而网格和云计算模型均支持软件和硬件的异构资源聚合调用。

2. 虚拟化

Web 服务没有虚拟化，提供的是系统的功能模块，网格和云计算分别支持虚拟化的技术，并且云计算是对硬件资源、操作平台的虚拟化，而网格只是数据和计算资源的虚拟化。

3. 应用驱动

Web 服务用户通过调用服务提供者暴露给外界的 API，使用该系统需要的某个特定功能。网格计算利用网络未用计算资源进行科学计算，云计算则提供给普通用户需要的各种服务，如存储、计算、应用服务等，具有更宽泛的适用性。

4. 可扩展性

Web 服务扩展能力有限，网格服务主要通过增加节点来扩展处理能力。云计算可根据需求，重新动态自动配置资源池，具有较好的扩展性。

5. 标准化

Web 服务和网格技术经过不断的发展和成熟，在用户调用以及内部资源调用接口上，

实现了较好的互操作性，而云计算由于本身发展的不完善性，在这方面还存在很多问题有待解决，制约了云计算的应用。

6. 节点操作系统

Web 服务和网络各节点都采用相同的操作系统，而云计算则比较灵活，提供了多种操作系统的虚拟机，为上层的云计算应用服务。

7. 容错性

云计算在实现机制上采取了冗余的数据副本，保证了不必像 Web 服务和网络计算那样数据执行失效后还要重新执行。

表 2.2 Web 服务、网络、云计算的比较

特征	Web 服务	网络	云计算
异构性	支持软件层次的异构性	支持软件、硬件层次的异构性	支持软件、硬件层次的异构性
虚拟化	无	数据和计算资源虚拟化	硬件、软件资源虚拟化
可扩展性	可变	可变，较好	按需提供
应用驱动	调用其他系统特定的功能模块	有限的科学计算服务	提供普通用户硬件、存储、软件等服务
标准化	比较完善	比较完善	有待解决
节点操作系统	相同的系统	相同的系统	多种操作系统的虚拟机
容错性	重新执行	重新执行	转移到其他节点继续执行

2.2.4 云计算应用分类

云计算的类型从不同的角度有不同的划分，本节在横向上按部署方式，在纵向上按云计算从底层到高层提供服务的方式分类介绍各种云计算，结合典型的云计算服务平台，由此在图 2.2 中分析云计算框架的构成，讨论各层次需要构建的机制和实现方案。

从云计算部署的角度，云计算分为私有云、社区云、公共云和混合云。私有云被一个组织管理操作。社区云由多个组织共同管理操作，具有一致的任务调度和安全策略。公共云由一个组织管理维护，提供对外的云服务，可以被公众所拥有。混合云是以上两种或两种以上云的组合。从云计算服务的角度，云计算服务类型可以分为基础设施即服务（Infrastructure as a Service, IaaS）、平台即服务（Platform as a Service, PaaS）、软件即服务（Software as a Service, SaaS）。

① IaaS 在服务层次上是底层服务，接近物理硬件资源，通过虚拟化的相关技术，为用户提供计算、存储、网络以及其他资源方面的服务，以便用户能够部署操作系统和运行软件。这一层典型的服务如亚马逊的弹性云（Amazon, EC2）。EC2 与 Google 提供的云计算服务不同，Google 只为互联网上的应用提供云计算平台，开发人员无法在这个平台上工作，因此只能转而通过开源的 Hadoop 软件支持来开发云计算应用。而 EC2 给用户提供一个虚拟的环境，使得可以基于虚拟的操作系统环境运行自身的应用程序。同时，用户可以创建

亚马逊机器镜像（AMI），镜像包括库文件、数据和环境配置，通过弹性计算云的网络界面去操作在云计算平台上运行的各个实例（Instance），同时用户需要为相应的简单存储服务（S3）和网络流量付费。

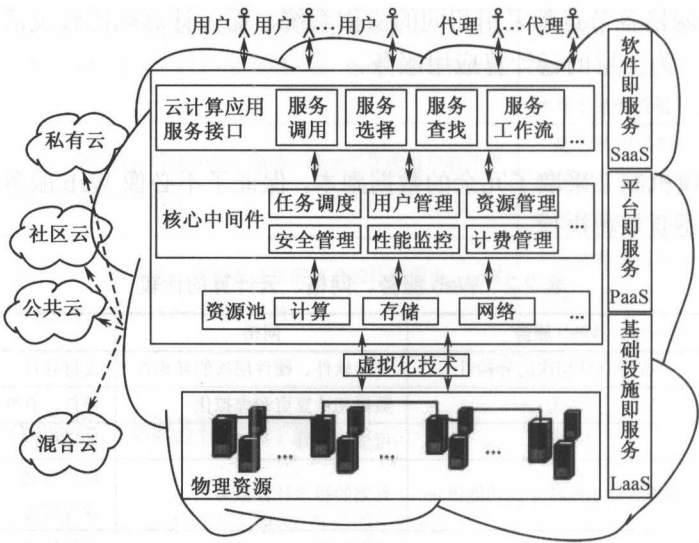


图 2.2 云计算框架图

② PaaS 是构建在基础设施即服务之上的服务，用户通过云服务提供的软件工具和开发语言，部署自己需要的软件运行环境和配置。用户不必控制底层的网络、存储、操作系统等技术问题，底层服务对用户是透明的，这一层服务是软件的开发和运行环境。这一层服务是一个开发、托管网络应用程序的平台，代表性的有 Google App Engine 和 Microsoft Azure。使用 Google App Engine，用户将不再需要维护服务器，用户基于 Google 的基础设施上传、运行应用程序软件。目前，Google App Engine 用户使用一定的资源是免费的，如果使用更多的带宽、存储空间等需要另外收取费用。Google App Engine 提供一套 API 使用 Python 或 Java 来方便用户编写可扩展的应用程序，但仅限 Google App Engine 范围的有限程序，现存很多应用程序还不能很方便地运行在 Google App Engine 上。Microsoft Azure 构建在 Microsoft 数据中心内，允许用户应用程序，同时提供了一套内置的有限 API，方便开发和部署应用程序。此平台包含在线服务 Live Service、关系数据库服务 SQL Services、各式应用程序服务器服务 NET Services 等。

③ SaaS 是前两层服务所开发的软件应用，不同用户以简单客户端的方式调用该层服务，例如以浏览器的方式调用服务。用户可以根据自己的实际需求，通过网络向提供商定制所需的应用软件服务，按服务多少和时间长短支付费用。最早提供该服务模式的是 Salesforce 公司运行的客户关系管理（CRM）系统，它是在该公司 PaaS 层 force.com 平台之上开发的 SaaS。Google 的在线办公自软件如文档、表格、幻灯片处理也采用 SaaS 服务模式。

云计算提供的不同层次服务使开发者、服务提供商、系统管理员和用户面临许多挑战。图 2.2 对此做出了归纳概述。底层的物理资源经过虚拟化转变为多个虚拟机，以资源池多重租赁的方式提供服务，提高了资源的效用。核心中间件起到任务调度、资源和安全管理、性能监控、计费管理等作用。一方面，云计算服务涉及大量的调用第三方软件及框架和重要数据处理的操作，这需要有一套完善的机制，以保证云计算服务安全有效地运行；另一方面，虚拟化的资源池所在的数据中心往往电力资源耗费巨大，解决这样的问题需要设计有效的资源调度策略和算法。在用户通过代理或者直接调用云计算服务的时候，需要和服务提供商之间建立服务等级协议（Service Level Agreement, SLA），那么必然需要服务性能监控，以便设计出比较灵活的付费方式。此外，还需要设计便捷的应用接口，方便服务调用。而用户在调用中选择什么样的云计算服务，这就要设计合理的度量标准并建立一个全球云计算服务市场以供选择调用。

2.2.5 小结

云计算是基于多种技术的新兴计算模式，随着现代软件应用和商务处理的全球化、信息化和自动化，必将为云计算的研究发展提供广泛的市场和应用背景。云计算不仅是虚拟化资源的集合，也不仅是在此之上的平台和应用实体的集合，而且是一种集虚拟化技术、网络技术、信息安全、效用计算、逻辑推理、软件工程、商务智能等技术为一体的新兴计算应用模式。无论是工业界还是学术界都提出了一系列实施技术和改进策略，并从理论和实际应用的角度进行了阐述。

由上面的讨论分析可知，应用向云计算模式的转变引发了一系列开放的问题，有待解决。

① 用户在选择使用众多云计算服务时，如何选择需要的服务应用，通过什么标准度量云计算服务特征，避免选择的主观性。

② 以往 Web 服务定义的 WSDL 接口和 XML 数据类型方便用户的调用和信息的传输，需要考虑云计算的接口，数据类型怎样制定，采取何种具体的标准加强云计算供应商和用户间的互操作尚不明确。

③ 随着云计算模式的大量应用，是否所有的软件应用和开发都适合转向云计算的平台，这就需要考虑建立软件应用属性到云计算服务属性的映射，以判定云计算的属性是否适合软件应用的关键属性。

④ 如何划分 SaaS 层次上云计算基本服务粒度，以便应用能够进行类似 Web 服务编排的服务组合，提高软件的重用性。

⑤ 云计算是一种分布式的计算模式，其地理位置、存储和扩展能力对用户均是透明的。无论是云计算开发者、提供商还是用户，如何追踪分析云计算服务应用的控制流和数据流，以判定云计算应用的行为和状态，是问题的关键。建立何种合适的模型，使云计算模型标准化、统一化，为测试、成本计算、性能提供标准依据，也是一个重要的问题。

⑥ 如何发挥云计算的特性，探索其在传统软件测试领域的应用。例如，传统软件测试

用例的运行需要耗费大量物理设备和时间,尤其是回归测试更需要不断地运行测试用例。如果把这些需要运行的测试用例转向云计算的环境并行运行,其效率会极大提高。

⑦ 云计算服务本身的质量关系到用户能否大量使用服务。目前,研究主要集中在底层基础设施服务的性能分析、优化以及测试研究,但是上层的云计算应用服务的测试模型和测试标准同样需要研究和关注。

2.3 大数据与云计算的关系

云计算技术自 2007 年以来得到了蓬勃的发展。云计算的核心模式是大规模分布式计算,将计算、存储、网络等资源以服务的模式提供给多用户,按需使用。云计算为企业和用户提供高可扩展性、高可用性和高可靠性,提高资源使用效率,降低企业信息化建设、投入和运维成本。随着美国亚马逊、Google、微软公司提供的公共云服务的不断成熟与完善,越来越多的企业正在往云计算平台上迁移。

由于国家的战略规划需要和政府积极引导,云计算及技术在我国近几年来取得了长足的发展。我国设立了北京、上海、深圳、杭州、无锡作为第一批云计算示范城市,北京的“祥云”计划、上海的“云海”计划、深圳的“云计算国际联合实验室”、无锡的“元云计算项目”,以及杭州的“西湖云计算公共服务平台”先后启动和上线,其他城市如天津、广州、武汉、西安、重庆、成都等也都推出了相应的云计算发展计划或成立了云计算联盟,积极开展云计算的研究开发和产业试点。然而中国云计算的普及在很大程度上仍然局限在基础设施的建设方面,缺乏规模性的行业应用,没有真正实现云计算的落地。物联网及云计算技术的全面普及是我们的美好愿景,能够实现信息采集、信息处理,以及信息应用的规模化、泛在化、协同化。然而其应用的前提是大部分行业、企业在信息化建设方面已经具备良好的基础和经验,有着迫切的需求去改造现有系统架构,提高现有系统的效率。而现实情况是我们的大部分中小企业在信息化建设方面才刚刚起步,只有一些大型企业和国家部委在信息化建设方面具备基础。

大数据的爆发是社会和行业信息化发展中遇到的棘手问题。由于数据流量和体量增长迅速,数据格式存在多源异构的特点,而我们对数据处理又要求能够准确实时,能够帮助我们发掘出大体量数据中潜在的价值。传统的信息技术架构,已无法处理大数据问题,它存在着扩展性差、容错性差、性能低、安装部署及维护困难等诸多瓶颈。由于物联网、互联网、移动通信网络技术在近些年的迅猛发展,造成数据产生和传输的频度和速度都大大加快,催生了大数据问题,而数据的二次开发、深度循环利用则让大数据问题日益突出。

我们认为云计算与大数据是相辅相成、辩证统一的关系。云计算、物联网技术的广泛应用是我们的愿景,而大数据的爆发则是发展中遇到的棘手问题。前者是人类文明追求的梦想,后者是社会发展亟待解决的瓶颈。云计算是技术发展趋势,大数据是现代信息社会飞速发展的必然现象。解决大数据问题,又需要现代云计算的手段和技术。大数据技术的

突破不仅能解决现实困难，同时也会促使云计算、物联网技术真正落地，并深入推广和应用。

从现代 IT 技术的发展中，我们能总结出几个趋势和规律。

① 大型机与 PC 之争，以 PC 完胜为终结。苹果 iOS 和 Android 之争，开放的 Android 平台在 2、3 年内即抢占了 1/3 的市场份额。Nokia 的塞班操作系统因为不开放，已经处于淘汰边缘。这些都体现了现代 IT 技术需要本着开放、众包的理念，才能取得长足发展。

② 现有的常规技术同云计算技术的碰撞与之相类似，云计算技术的优势在于利用众包理论和开源体系，建设基于开放平台和开源新技术的分布式架构，能够解决现有集中式的大机处理方式难以解决或不能解决的问题。像淘宝、腾讯等大型互联网公司也曾经依赖于 Sun、Oracle、EMC 这样的大公司，后来都因为成本太贵而采用开源技术，自身的产品最终也贡献给开源界，这也反映了信息技术发展的趋势。

③ 传统行业巨头已经向开源体系倾斜，这是利于追赶的历史机遇。传统的行业巨头、大型央企如国家电网、电信、银行、民航等因为历史原因过度依赖外企成熟的专有方案，造成创新性不足，被外企产品绑架的格局。从破解问题的方案路径上分析，解决大数据问题，必须逐渐放弃传统信息技术架构，利用以云技术为代表的新一代信息技术来解决大数据问题。尽管先进的云计算技术主要发源于美国，但是基于开源基础，我们与发达技术的差距并不大，将云计算技术应用于大型行业中的迫切的大数据问题，也是我们实现创新突破、打破垄断、追赶国际先进技术的历史契机。

2.3.1 大数据是信息技术发展的必然阶段

根据今天的信息技术的发展情况，我们预测：各个国家和经济实体，都会将数据科学纳入亟待研究的应用范畴，数据科学将发展成为人类文明中一门至关重要的宏观科学，其内涵和外延已经覆盖所有同数据相关的学科和领域，逐渐构架出清晰的纵向层级关系和横向扩展边界。

纵向上，从文字、图像的出现算起，发展到以数学为基础的自然学科，再发展到以计算机为工具，甚至到云计算、物联网、移动互连的今天，围绕的核心就是数据。只是今天的数据，按照我们的宏观数据理论，已经扩展为所有人类文明所记载的内容，而不再是狭义的数值。

横向上，数据科学正向其他社会学科和自然学科渗透，并很大程度地影响了其他学科研发流程和探究方法的传统思维，建立了各个学科、各个领域间的新型关联关系，弱化了物理性边界，使事物和事件变得更加一体化。

正是这种横向、纵向上的延展，使数据的包容性达到了前所未有的数量、容量和质量，而且加速倾向严重，其重要性更是上升到了生产要素的战略高度，使人们意识到大数据时代（或叫数据时代）真正来临了。这一切的起因，就是信息技术的高速发展。

所以说，大数据是我们必须面临的问题，是我们发展中必然要经历的阶段。

2.3.2 云计算等新兴信息技术正在真正地落地和实施

国内云计算及大数据市场已经具备初步发展态势。2010年,中国云计算市场规模同比增长29.3%。计世资讯研究表明,在企业用户中,已经有67.5%的用户认可云服务模式,并开始采用云计算服务,或者在企业内部实现云平台共享。市场规模也从2010年的167.31亿元增长到2013年的1174.12亿元,年均复合增长率达到91.5%。未来几年云计算应用将以政府、电信、教育、医疗、金融、石油石化和电力等行业为重点发展。

云计算及大数据处理技术已经渗透到国内传统行业及新兴产业,政策、资金引导力度不断加大。纵观国内市场,云计算已广泛应用在互联网企业、社交网站、搜索、媒体、电子商务等新兴产业领域。同时,在国家的政策引导下,科研经费投入力度加大,国家重大项目资金、政府引导型基金、地方配套资金和企业发展所需的科研基金涉及了国民经济多个支柱型行业和领域,其规模、数量增长迅猛,时效显著。在这一大背景下,传统行业的云计算应用将蓬勃发展起来,但目前大多仍着眼于硬件建设和资源服务层面(如智慧城市中宽带建设、数据中心项目等),核心软件关键技术如大数据处理方面,更多的是在课题研究领域,真正的应用也不多见。

重点领域的行业需求迫切。可以看到的是,这种市场状况正在改善,首先是一些企业(电力、民航、银行、电信)为了自身业务的发展需要,确实迫切需要新的技术解决在大数据处理方面所遇到的问题;其次,随着经济的高速发展以及市场环境的不断变化,越来越多的企业意识到了数据在开拓市场、提升自身竞争力等方面所起到的重要作用,挖掘数据、寻找新价值的需求逐渐受到了重视。同时,现代信息技术作为产业升级、打造新兴产业的引擎,又极大地推动了大数据处理技术的发展。可以预见大数据处理市场将会变得空前广阔,数据为王的理念将会被越来越多的人所接受。

2.3.3 云计算等新兴技术是解决大数据问题的核心关键

云计算等新兴信息技术诞生的初衷,是解决原有信息技术的高成本和高含量这个弊端。这个弊端经常让使用者用不起,搞不懂,影响信息技术的应用和创新。但云计算的迅速崛起,逐步解决了高成本、高含量的问题,但低成本、高速度的数据应用,也使数据泛滥成灾,出现数量大、结构变化快、速度时效性高、价值密度低等几大问题,促成了大数据这个概念。只有解决大数据这个疑难杂症,才能使云计算等新兴技术真正落地和实施。怎么解决、用什么技术、坚持什么原则,是需要我们认真考虑的问题。

大数据问题的解决,首先要从大数据的源头开始梳理。既然大数据源于云计算等新兴IT技术,就必然有新兴IT技术的基因继承下来。低成本、按需分配、可扩展、开源、泛在化等特点是云计算的基因,这些基因体现在大数据上时,有了性质的突变。如低成本这个基因,在大数据问题上就演变出数据产生的低成本和数据处理的高成本,按需分配的虚拟

化基因，促使数据的应用变得更加平台集中化，可扩展、开源和泛在化使数据变得增速异常等。综合起来就是：大量的普遍存在的低成本、低价值密度数据，多集中在平台上，使我们处理成本加大，技术难度加大，而且泛在化倾向加重。

泛在化倾向的加重，就意味着这个问题本身是全链条、全领域的增速共生事件，就必须以最广泛的视野和观念来克服和改善，简单的单项处理技术和局部突破在这个数据裂变量面前经常会变得力不从心，无法完成。这同云计算技术突破传统 IT 技术的大机原理、高成本瓶颈和技术垄断是一个道理。这说明低成本的复制、可扩展的弹性、众人参与的开源等原则既是云计算的基础手段，也是解决大数据问题的最实用的办法。再深入分析，云计算等先进的 IT 技术，天性就是要快速、方便、便宜地解决数据，所以，“解铃尚需系铃人”的逻辑思维是我们最便捷的解决路径，特别是互联网产业的爆炸式发展，让这个路径变得越来越唯一。覆盖和变革全信息产业的云计算等新兴 IT 技术，抽象出了“云”的理念、原则和手段，成为了我们理解大数据、克服大数据、应用大数据的制胜法宝和关键。

2.4 云资源调度与管理

建立起云计算数据中心和应用平台后，一项重要和关键的技术是如何将云计算数据中心虚拟共享资源有效地按用户需求动态管理和分配，并提高资源的使用效率从而为云计算的广泛应用提供便利。这其中涉及两个技术点：数据中心的资源调度和管理。图 2.3 就是资源调度、管理流程的一个示例。

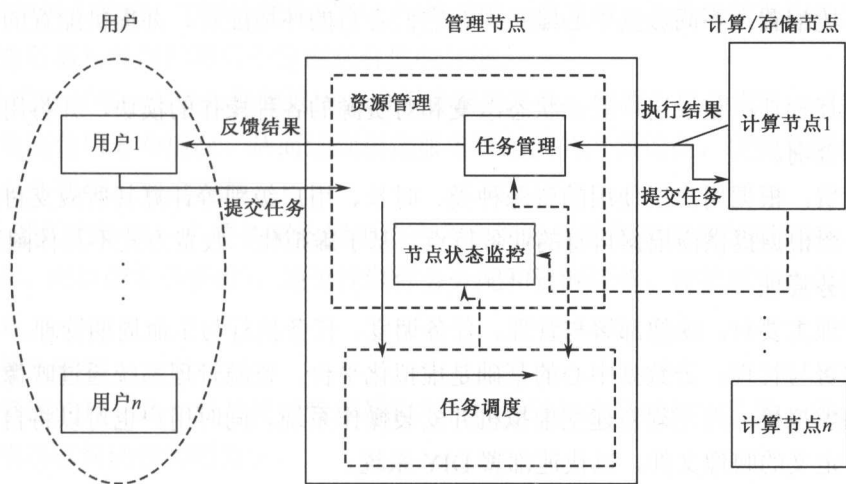


图 2.3 基于集群计算的调度、管理流程

本节主要介绍云计算数据中心资源调度与管理的基本概念，以及关键的技术点。

2.4.1 云资源管理

在数据中心规模日益庞大的今天,如果不能提升数据中心的**管理能力、全面充分地调度数据中心各项资源,那么这样的数据中心在性能上并不能称得上优秀,特别是服务器数量增加、虚拟化环境日趋复杂、数据中心能耗增加对数据中心管理者在服务器利用、服务器能耗等方面提出了极大挑战。因此,只有采用更加高效的数据中心管理平台,才能让数据中心的性能更上一个台阶。

对数据中心的**管理要从3方面入手,第一步就是搭建最基础的数据中心设备管理平台,通过这个平台对数据中心内部的各个设备进行实时的监控,当出现异常情况后,立即通过管理软件对其进行处理。第二步就是管理和控制能源消耗的设备,以及对已经部署的制冷设备进行实时调节。第三步则是对虚拟层设备的管理,主要是对实施虚拟化后设备的运行情况进行监视,以免因虚拟层的崩溃而对设备的正常运行造成影响。

1. 云数据中心资源管理的内容

云数据中心资源管理的内容主要为**用户管理、任务管理与资源管理。

(1) 用户管理

用户管理主要分为:账号管理、用户环境配置、用户交互管理与使用计费。

账号管理:云数据中心的主要作用之一就是为用户提供计算和存储资源。使用这些资源,用户应当注册账号以便于统一管理。同样地,数据中心管理员登录高权限的账号,可以对数据中心进行普通用户无法访问的操作。

用户环境配置:不同数据中心账户保存它们各自的环境配置,并提供配置的导出和导入功能。

用户交互管理:记录用户登录状态改变和对资源的各种操作的模块,并将用户操作写入日志以备查询。

使用计费:根据用户所使用的资源种类、时长、用户级别等计算其所应支付的费用,计费系统一般根据提供商根据自身的业务特点,基于虚拟化。收费方式不具体阐述。

(2) 任务管理

任务管理主要有:映像部署与管理、任务调度、任务执行与生命周期管理。

映像部署与管理:云数据中心的基础是虚拟化平台,资源管理系统通过映像文件部署一台全新的虚拟机,而无须新建空虚拟机并安装操作系统。同时用户也可以将自己的虚拟机保存为自定义的映像文件,以快速部署DIY系统。

任务调度:负责在数据中心服务器上分配用户任务的模块。

任务执行:负责执行数据中心具体的任务的模块。

生命周期管理:对资源生命周期进行管理,定期释放过期的资源以节省数据中心存储空间和能耗。

(3) 资源管理

资源管理主要内容为：多种调度算法、故障检测、故障恢复与监控统计。

多种调度算法：负责从监控统计模块获取数据，计算数据中心各个服务器的负载状态，并在合适的时候执行多种调度算法，以使所有的服务器工作在最佳的状态。

故障检测：该模块周期性地启动，测试数据中心的软硬件状况，记入日志或者数据库，并且在检测到指定错误时向管理员报告。

故障恢复：通常对可预计的故障预先设定好故障处理模块，当发生这些故障时，将会自动启动应对措施。

监控统计：监控数据中心各类资源的状态，汇总数据并及时提供给其他模块进行相应的计算。

2. 资源管理的目标

云计算的资源管理的目标就是接受用户的资源请求，并把特定的资源分配给资源的请求者，主要包括数据存储和资源管理两个方面的内容。在这里，我们将云资源管理的目标概括为以下几点。

(1) 自动化

自动化就是数据中心资源管理模块在无须人工干预的情况下能够处理用户请求、服务器软、硬件故障以及对各项操作进行记录。

(2) 资源优化

定时对数据中心资源分配进行优化，以保持数据中心资源的合理分配。资源的优化依据不同的策略，不同的策略有不同的优化目标，通常有以下几种。

① 通信调优策略：主要依据数据中心网络带宽调度资源，该策略使得服务器之间的通信带宽、服务器与外部的通信带宽得到合理的分配。

② 热均衡策略：主要依据数据中心内服务器的产热分布进行资源调度，该策略调整数据中心的资源使用分布情况，从而达到指定服务器之间的产热均衡，使得数据中心的散热设备得到充分利用，节约资源。

③ 负载均衡策略：主要依据数据中心内各个服务器的物理资源使用情况（主要包括CPU、内存、网络带宽等资源），通过控制任务分配和资源迁移，使数据中心达到综合负载均衡的状态。

(3) 简洁管理

资源管理的目标之一就是使得管理员和用户能够较为容易地管理资源，因此，功能和界面设计应当以简洁和实用为主。

(4) 虚拟资源与物理资源的整合

虚拟资源与物理资源的整合是通过虚拟化技术实现的，虚拟化技术对于创建云计算中心至关重要。虚拟化技术是云计算中的一个关键技术，因为云计算中一台主机能够同时运行多个操作系统平台，其处理能力和存储空间也能根据需求不同而被不同平台上的应用动

态共享。动态的分配和回收物理主机资源，很大程度上增加了云资源管理的难度。

2.4.2 云资源调度策略

1. 资源调度关键技术

云计算建立在计算机界长期的技术累计基础之上，包括软件 and 平台作为一种服务、虚拟化技术和大规模的数据中心技术等关键技术。数据中心（可能是分布在不同地理位置的多个系统）是容纳计算设备资源的集中之地，同时负责对计算设备的能源提供和空调维护等。数据中心可以单独建设，也可以置于其他建筑之内。动态分配管理虚拟和共享资源在新的应用环境——云计算数据中心里面新的挑战，因为云计算应用平台可能分布广泛而且种类多样，加之用户需求的实时动态变化很难准确预测，以及需要考虑系统性能和成本等因素使得问题非常复杂。需要设计高效的云计算数据中心分配调度策略算法以适应不同的业务需求和满足不同的商业目标。目前的数据中心分配调度策略主要包括先来先服务、负载均衡、最大化利用等。提高系统性能和服务质量是数据中心的关键技术指标，然而随着数据中心规模的不断扩大，能源消耗成为日益严重和备受关注的问题，因为能源消耗对成本和环境的影响都极大。

云数据中心资源调度关键技术主要包括以下几个方面。

① 调度策略：是资源调度管理的最上层策略，需要数据中心所有者和管理者界定，主要是确定调度资源的目标，以及确定当资源不足时满足所有立即需求时的处理策略。

② 优化目标：调度中心需要确定不同的目标函数以判断调度的优劣，目前有最大化满足用户请求、最低成本、最大化利润、最大化资源利用率等优化目标函数。

③ 调度算法：好的调度算法需要按照目标函数产生优化的结果，并且在极短的时间之内，同时自身不能消耗太多资源。一般来讲，调度算法基本都是 NP-Hard 问题，需要极大的计算量而且不能通用。业界普遍采用近似优化的调度算法并且针对不同应用调度算法不同。

④ 调度系统结构：与数据中心基础架构密切相关，目前多是多级分布式体系结构。

⑤ 数据中心资源界定及其相互制约关系：分析清楚资源以及其相互制约关系有利于调度算法综合平衡各类因素。

⑥ 数据中心业务流量特征分析：掌握业务流量特征有助于优化调度算法。

2. 资源调度策略分类

1) 性能优先

(1) 先来先服务

最大限度地满足单台虚拟机的资源要求，一般采用先来先服务的策略，同时结合用户优先级。主要考虑如何最大化满足用户需求，并考虑用户优先级别（包括重要性和安全性等）。初期的 IBM 虚拟计算等都是如此，多用于公司或学校内部。可能没有具体的调度优

化目标函数，但须说明管理员是如何分配资源的。服务器可分为普通、高吞吐量、高计算密度等类别供用户选择。

(2) 负载均衡

负载均衡是指使所有服务器的平均资源利用率达到平衡，如 VMware 和 Sun 公司产品等采用了负载均衡策略。

优化目标：资源利用的平衡，即所有物理服务器（CPU、内存利用率、网络带宽等）利用率基本一致。

每当有资源被分配使用时，需要计算、监控各资源目前的利用率（或者直接使用负载均衡分配算法），将用户分配到资源利用率最低的资源上。

通过软、硬件都可以实现。硬件方式通过提供负载平衡专门的设备，如多层交换机，可以在一个集群内分发数据包。通常情况下，实施、配置和维护基于硬件的解决方案需要时间和资金成本的投资。软件方式可以采用 Round Robin 等调度方式。

(3) 提高可靠性

优化目标：使各资源的可靠性达到指定的具体要求（例如保证业务 99.9% 时间）。例如，Amazon 99.95% 的业务可靠性承诺。

业务可靠性与服务器本身的可靠性（平均故障时间、平均维修时间等）相关，还有停机、停电、动态迁移等造成的业务中断将影响业务的可靠性。

例如一台物理服务器的可靠性是 90%，用户要求的业务的可靠性是 99.9%，调度需要至少双机备份。假设一次动态迁移对于业务的可靠性降低 0.1%，则调度策略需要减少（或避免）动态迁移。

在一定前提下，尽量减少虚拟机迁移次数（平均迁移次数、总迁移次数、单台虚拟机最大迁移次数）。需要统计虚拟机迁移对可靠性造成的量化影响。

提高可靠性的方式是备份冗余等方式，使用主备份方式时主用机与备用机不放置在同一物理机上或同一机架上。具体指标也可以由用户指出（作为需求选项由用户选择）。

2) 成本优先

(1) 提高整体利用率

优化目标：资源利用率最大，使所有数据中心计算资源得到最大程度的利用（或用最少的物理机满足用户需求）。

输入：当前数据中心的资源分布，用户请求（特定的虚拟机）。

输出：用户请求的虚拟机配置在数据中心的物理机编号。

定义：

物理（虚拟）服务器的利用率（或效率）= 已分配 CPU / 已开物理机可虚拟出的 CPU 总数

这一参数说明当前服务器的使用情况，由此可以排列出不同服务器效率的高低。选择虚拟机时总是按照其利用率从小到大排列。

每台虚拟机单位时间内的价格 = 虚拟机在单位时间的成本 $\times (1+a)$

其中： a 为利用率，可由提供商控制。

虚拟机单位时间内的成本可由其占用的计算资源、存储资源和网络资源的成本进行估算（取较大值）。

（2）最大化利润

优化目标：最大化利润，使用各种资源的收入（单位时间）减去使用各种资源的总成本得出利润。

考虑因素主要包括以下几点。

① 单位资源单位时间的成本（每台物理机可能不一样）=固定成本（含折旧、人力等）+变动成本（与其功耗相关），虚拟机的功耗率=虚拟机满负载的总成本/虚拟机总 CPU 容量。

② 每台物理机上的成本=启动成本（每次新开一台服务器的成本）+单位资源单位时间的成本×时间×资源大小。

③ 单个用户请求的收入=该用户选择的虚拟机单位时间价格×使用时间，资源总收入为所有用户的收入之和。

④ 每次用户使用结束后，比较迁移条件，如果满足则可以进行迁移，以减少物理服务器开机数量，减少成本。

（3）最小化运营成本

最大限度降低运营成本，减少制冷、电力、空间成本。

优化目标：最小化成本，使所有资源成本之和最小化。考虑因素主要包括以下几点。

① 单位资源单位时间的成本（每台物理机可能不一样）=固定成本（含折旧、人力等）+变动成本（与其功耗相关）。

② 虚拟机的功耗率=虚拟机满负载的总成本/虚拟机总 CPU 容量。

③ 每台物理机上的成本=启动成本（每次新开一台服务器的成本）+单位资源单位时间的成本×时间×资源大小。

④ 单个用户请求的收入=该用户选择的虚拟机单位时间价格×使用时间，资源总收入为所有用户的收入之和。

综上所述，需要考虑到公司实际的业务需求和商业目标而选取不同的调度策略。对于满足公司内部业务需求为主的应用，可以考虑最小化成本、最大化利用率和负载均衡等；对于商业应用为主的需求，可能考虑最大化利润较好。

2.4.3 云计算数据中心负载均衡调度

本节主要介绍云计算数据中心资源负载均衡调度算法。

1. 云计算数据中心综合负载均衡调度策略概述

云计算数据中心将虚拟机按用户需求规格（可能不一致）动态、自动化地分配给用户，但是由于用户的需求规格和数据中心所有的物理服务器的规格配置不一致，如果采用简单的分配调度方法，例如常用的轮转法、加权轮转法、最小负载（或链接数）优先、加权最

小负载优先法、哈希法等，很难达到物理服务器负载均衡，进而会造成服务性能不均衡和其他相关问题。

轮转法（Round Robin）通常是预先设定好一个轮转周期（例如物理服务器个数），依次将用户需求的虚拟机分配给不同的物理服务器，一个轮转周期结束后重新开始新一轮转。轮转法不能解决物理服务器和用户需求规格不一致造成的负载不均衡问题。

加权轮转法预先对物理服务器设定权值，在负载均衡分配虚拟机的过程中，轮转选择物理服务器，如果被选择的物理服务器的权值为 0，则跳过该服务器并选择下一台，如被选择的服务器的权值不为 0，则选中该服务器并将该服务器的权值减 1，后继的选择在前一次选择的基础上轮转。以权值分别为 1, 2, 3 的 3 台物理服务器（ PM_1 , PM_2 , PM_3 ）为例，第一次选择第一台物理服务器 PM_1 ，其权值减为 0，第二次选择第二台物理服务器 PM_2 ，其权值减为 1，第三次选择第三台物理服务器 PM_3 ，其权值减为 2，第四次轮转到第一台服务器 PM_1 ，但是其权值为 0，继续轮转，选择第二台服务器 PM_2 ，同时其权值减为 0，依此类推。6 次选择依次是： PM_1 , PM_2 , PM_3 , PM_2 , PM_3 , PM_3 。这样权值高的服务器获得的服务次数就与其权值成正比，但是当用户需求规格不一致时仍然存在负载不均衡的问题；另外加权轮转法需要在均衡过程中修改各台服务器的权值，这些公共变量需要进行加锁、解锁，影响执行速度。

2. 云计算数据中心负载均衡调度策略中主要调度算法分析

本节主要介绍的调度算法包括：轮转调度算法、加权轮转调度算法、目标地址哈希调度算法、源地址哈希调度算法、加权最小链接算法。

（1）轮转调度算法

把新的连接请求按顺序轮流分配到不同的服务器上，从而实现负载均衡。该算法的优点在于简单易行，但不适用于每个服务器性能不一致的情况。

轮转调度算法（Round Robin Scheduling）就是以轮转的方式依次将请求调度到不同的服务器，即每次调度执行 $i = (i+1) \bmod n$ ，并选出第 i 台服务器。算法的优点是简洁，无须记录当前所有连接的状态，所以它是一种无状态调度。

在系统实现时，引入了一个额外条件，当服务器的权值为零时，表示该服务器不可用而不被调度。这样做的目的是将服务器切出服务（如屏蔽服务器故障和系统维护），同时与其他加权算法保持一致。所以，算法要做相应的改动，它的算法流程如下。

轮转调度算法流程：假设有一组服务器 $S = \{S_0, S_1, \dots, S_{n-1}\}$ ，一个指示变量 i 表示上一次选择的服务器， $W(S_i)$ 表示服务器 S_i 的权值。变量 i 被初始化为 $n-1$ ，其中 $n > 0$ 。

```

j = i;
do{
    j = (j+1) mod n;
    if(W(Sj) > 0) {
        i = j;
    }
}

```



```

        return Si;
    }
} while(j != i);
return null;

```

轮转调度算法假设所有的服务器处理性能均相同，不管服务器的当前连接数和响应速度。该算法相对简单，不适用于服务器组中处理性能不一致的情况。

(2) 加权轮转调度算法

克服轮转调度算法的不足，用相应的权值表示服务器的处理能力，权值较大的服务器将被赋予更多的请求。一段时间后服务器处理的请求数趋向于各自权值的比例。

加权轮转调度算法流程：

假设有一组服务器 $S = \{S_0, S_1, \dots, S_{n-1}\}$ ， $W(S_i)$ 表示服务器 S_i 的权值，一个指示变量 i 表示上一次选择的服务器，指示变量 cw 表示当前调度的权值， $\max(S)$ 表示集合 S 中所有服务器的最大权值， $\gcd(S)$ 表示集合 S 中所有服务器权值的最大公约数。变量 i 初始化为 -1， cw 初始化为零。

```

while (true) {
    i = (i + 1) mod n;
    if (i == 0) {
        cw = cw - gcd(S);
        if (cw <= 0) {
            cw = max(S);
            if (cw == 0) {
                return NULL;
            }
        }
    }
    if (w(Si) >= cw)
        return Si;
}

```

加权轮转调度算法考虑了服务器处理性能不一致、服务器的当前连接数等因素。该算法相对轮转调度实用性更强，但是当请求服务时间变化比较大时，加权轮转调度算法容易导致服务器间的负载不平衡。

(3) 目标地址哈希调度算法

以目的地址为关键字查找一个静态哈希 (Hash) 表来获得所需的真实服务器。

目标地址哈希调度算法 (Destination Hashing Scheduling) 也是针对目标 IP 地址的负载均衡，但它是一种静态映射算法，通过一个哈希函数将一个目标 IP 地址映射到一台服务器。

目标地址哈希调度算法先根据请求的目标 IP 地址，作为哈希键（Hash Key）从静态分配的哈希表找出对应的服务器，若该服务器是可用的且未超载，将请求发送到该服务器，否则返回空。该算法的流程如下。

假设有一组服务器 $S = \{S_0, S_1, \dots, S_{n-1}\}$ ， $W(S_i)$ 表示服务器 S_i 的权值， $C(S)$ 表示服务器 S_i 的当前连接数。ServerNode[] 是一个有 256 个桶（Bucket）的哈希表，一般来说，服务器的数目会远小于 256，当然表的大小也是可以调整的。

算法的初始化是将所有服务器顺序、循环地放置到 ServerNode 表中。若服务器的连接数大于 2 倍的权值，则表示服务器已超载。

```
if(( n is dead) OR
(W(n) == 0) OR
(C(n) > 2*W(n)) then
return NULL;
return n.
```

在实现时，采用素数乘法 Hash 函数，通过乘素数使得哈希键值尽可能地达到较均匀分布，所采用的素数乘法 Hash 函数如下。

```
static inline unsigned hashkey(unsigned int dest_ip)
{
    return(dest_ip * 2654435761UL) & HASH_TAB_MASK;
}
```

其中，2654435761UL 是 2 到 2^{32} （4294967296）间接近于黄金分割的素数。

（4）源地址哈希调度算法

以源地址为关键字查找一个静态 Hash 表来获得所需的真实服务器。

源地址哈希调度算法（Source Hashing Scheduling）正好与目标地址哈希调度算法相反，它根据请求的源地址，作为哈希键（Hash Key）从静态分配的哈希表中找出对应的服务器，若该服务器是可用的且未超载，将请求发送到该服务器，否则返回空。它采用的哈希函数与目标地址哈希调度算法相同。它的算法流程与目标地址哈希调度算法基本相似，区别在于将请求的目标 IP 地址换成请求的源 IP 地址，所以这里不重复叙述。

在实际应用中，源地址哈希调度和目标地址哈希调度可以结合使用在防火墙集群中，它们可以保证整个系统的唯一出入口。

（5）加权最小链接算法

克服最小链接算法的不足，用相应的权值表示服务器的处理能力，将用户的请求分配给当前连接数与权值之比最小的服务器。它是 LVS（Linux Virtual System）默认的负载分配算法。假设有一组服务器 $S = \{S_0, S_1, \dots, S_{n-1}\}$ ， $W(S_i)$ 表示服务器 S_i 的权值， $C(S_i)$ 表示服务器 S_i 的当前连接数，所有服务器当前连接数的总和为 $C_{\text{sum}} = \sum_{i=1}^{n-1} C(S_i)$ 。当前的新连接请求

会被发送服务器 S_m ，当且仅当服务器满足以下条件：

$$\frac{C(S_m)}{C_{\text{sum}} W(S_m)} = \min \left\{ \frac{C(S_m)}{C_{\text{sum}} W(S_i)} \right\} (i = 0, 1, \dots, n-1)$$

其中 $W(S_i)$ 不为零因为 C_{sum} 在这一轮查找中是个常数，所以判断条件可以简化为

$$\frac{C(S_m)}{W(S_m)} = \min \left\{ \frac{C(S_i)}{W(S_i)} \right\} (i = 0, 1, \dots, n-1)$$

其中 $W(S_i)$ 不为零。

因为除法所需的 CPU 周期比乘法多，且在 Linux 内核中不允许浮点除法，服务器的权值大于零，所以判断条件 $\frac{C(S_m)}{W(S_m)} > \frac{C(S_i)}{W(S_i)}$ 可以进一步优化为 $C(S_m) \times W(S_m) > C(S_i) \times W(S_i)$ 。

同时保证服务器的权值为零时，服务器不被调度。所以算法只要执行以下流程：

```
for(m=0;m<n;m++) {
    if(W(Sm) > 0) {
        for(i=m+1; i<n; i++) {
            if(C(Sm)*W(Si) > C(Si)*W(Sm))
                m = i;
        }
        return Sm;
    }
}
```

2.5 开源云管理平台 OpenStack

大数据处理需要大规模物理资源的云数据中心和具备高效的调度管理功能的云计算平台的支撑。云计算平台能为大型数据中心及企业提供灵活高效的部署、运行和管理环境，通过虚拟化技术支持异构的底层硬件及操作系统，为应用提供安全、高性能、高可靠性和高伸缩性的云资源管理解决方案，降低应用系统开发、部署、运行和维护的成本，提高资源使用效率。

作为新兴的计算模式和商业模式，云计算在学术界和业界获得了巨大的发展动力，政府、研究机构和行业领跑者正在积极地尝试应用云计算来解决网络时代日益增长的计算和存储问题，诞生了如 OpenStack、OpenNebula、Eucalyptus、Nimbus 和 CloudStack 等开源云平台。另外，全球各大互联网公司也在极力打造自己的商业云平台，如亚马逊的 AWS、谷

歌的 AppEngine、阿里巴巴的阿里云和微软的 Windows Azure Services 等商业云计算平台相继出现，无论是开源的还是商业的，每个云计算平台都有显著的特点和不断发展的社区。

在所有开源云平台中，OpenStack 拥有最大的开源社区用户数和最高的社区活跃度，IBM、Intel、微软、思科、Dell、中国开源云联盟等都是 OpenStack 的成员单位。图 2.4 对四大开源云计算平台的社区活跃度做了对比。

OpenStack 既是一个社区，也是一个开源的云计算管理平台项目，由几个主要的组件组合起来完成具体工作。OpenStack 支持几乎所有类型的云环境，项目目标是提供实施简单、可大规模扩展、丰富、标准统一的云计算管理平台。OpenStack 通过各种互补的服务提供了基础设施即服务（IaaS）的解决方案，每个服务提供 API 以进行集成。

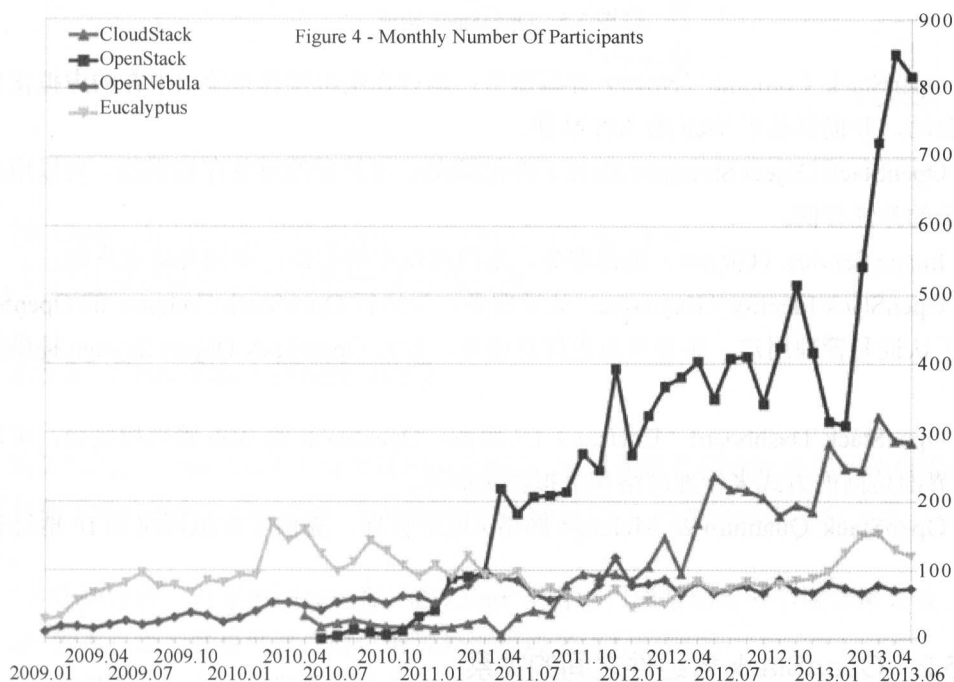


图 2.4 四大开源云计算平台社区活跃度对比

2.5.1 OpenStack 的构成

OpenStack 是一个完全开源的云计算系统，使用者可以在需要的时候修改代码来满足需要并作为开源或商业产品发布、销售；同时，OpenStack 基于强大的社区开发模式，任何公司和个人都可以参与到项目中，参与测试开发，贡献代码；目前，OpenStack 主要由六大组件构成，如图 2.5 所示。

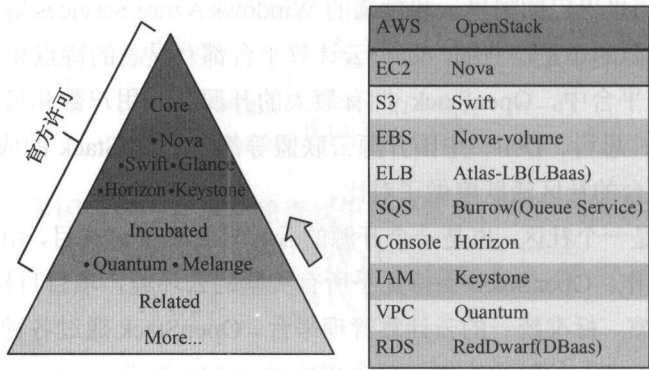


图 2.5 OpenStack 构成

- ① OpenStack Compute (Nova) 计算服务：运行在主机操作系统上潜在的虚拟化机制交互的驱动，并提供基于 Web 的 API 功能。
- ② OpenStack Object Storage (Swift) 存储服务：可扩展的对象存储系统，可以用来创建基于云的弹性存储。
- ③ Image Service (Glance) 镜像服务：虚拟机镜像的存储、查询和检索系统。
- ④ OpenStack Identity (Keystone) 认证服务：为运行 OpenStack Compute 的 OpenStack 云提供了认证和管理用户、账号和角色信息服务，并为 OpenStack Object Storage 提供授权服务。
- ⑤ OpenStack Dashboard (Horizon) UI 服务：OpenStack 的 Web 管理控制台，可以通过 Web 界面访问的方式来管理网络和虚拟机实例等。
- ⑥ OpenStack Quantum & Melange 网络&地址管理：提供了虚拟网络和 IP 地址管理服务。

2.5.2 OpenStack 各组件之间的关系

OpenStack 的设计目标是成为一个“可交付的大型可伸缩的云操作系统”。为了达到这个目标，每个组件、服务相互协作，共同提供一个完整的基础设施即服务（IaaS）。这种集成通过每个服务提供公共应用程序编程接口（API）来实现。因为这些 API 被用做服务与服务之间相互协调的方式，所以也允许底层的这些服务可以任意替换，而不会影响其他服务，因为与这些服务相互通讯的 API 永远不会变化。这些组件最终也都提供相同的 API 给云的终端用户。图 2.6 是 OpenStack 六大组件的逻辑关系图。

- ① Dashboard 提供了一个统一的 Web 操作界面来访问其他的 OpenStack 服务。
- ② Compute 通过 Image 存储和检索虚拟磁盘文件和相关元数据。
- ③ Network 为 Compute 提供了虚拟网络。
- ④ Block Storage 为 Compute 提供了存储卷。

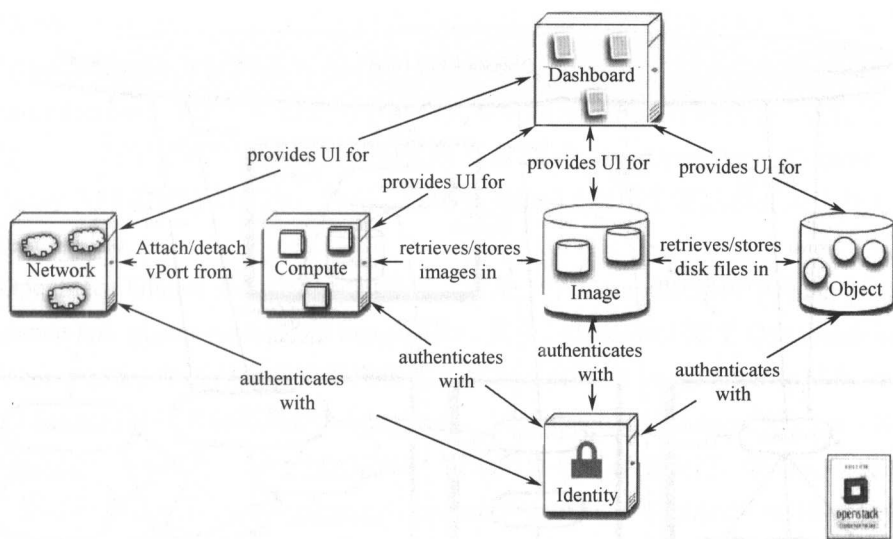


图 2.6 OpenStack 六大组件的逻辑关系

⑤ Image 可以将实际的虚拟磁盘文件存储到 Object Store 上。

⑥ 所有服务都要通过 keystone 来授权访问。

2.5.3 OpenStack 的逻辑架构

图 2.7 给出了 OpenStack 主要模块的一些细节，可以帮助我们更好地理解如何设计部署、安装和配置这个平台。模块根据其所属的功能组织起来，并根据类型进行分类。这些类型如下。

① 守护进程：以守护进程运行，在 Linux 平台上，通常作为一个服务来安装。

② 脚本：当一些事件发生时，通过外部模块来运行的脚本。

③ 客户端：一个访问服务所绑定的 Python 的客户端。

④ CLI：一个提交命令的命令行解释器。

下面针对图 2.7 所描述的逻辑结构进行阐述。

① 终端用户通过 nova-api 对话来与 OpenStack Compute 交互，通过 glance-api 对话来与 OpenStack Glance 交互，通过 OpenStack Object API 来与 OpenStack Swift 交互。

② OpenStack Compute 守护进程之间通过队列（行为）和数据库（信息）来交换信息，以执行 API 请求。

③ OpenStack Glance 与 OpenStack Swift 基本上都是独立的基础架构，OpenStack Compute 通过 Glance API 和 Object API 来进行交互。

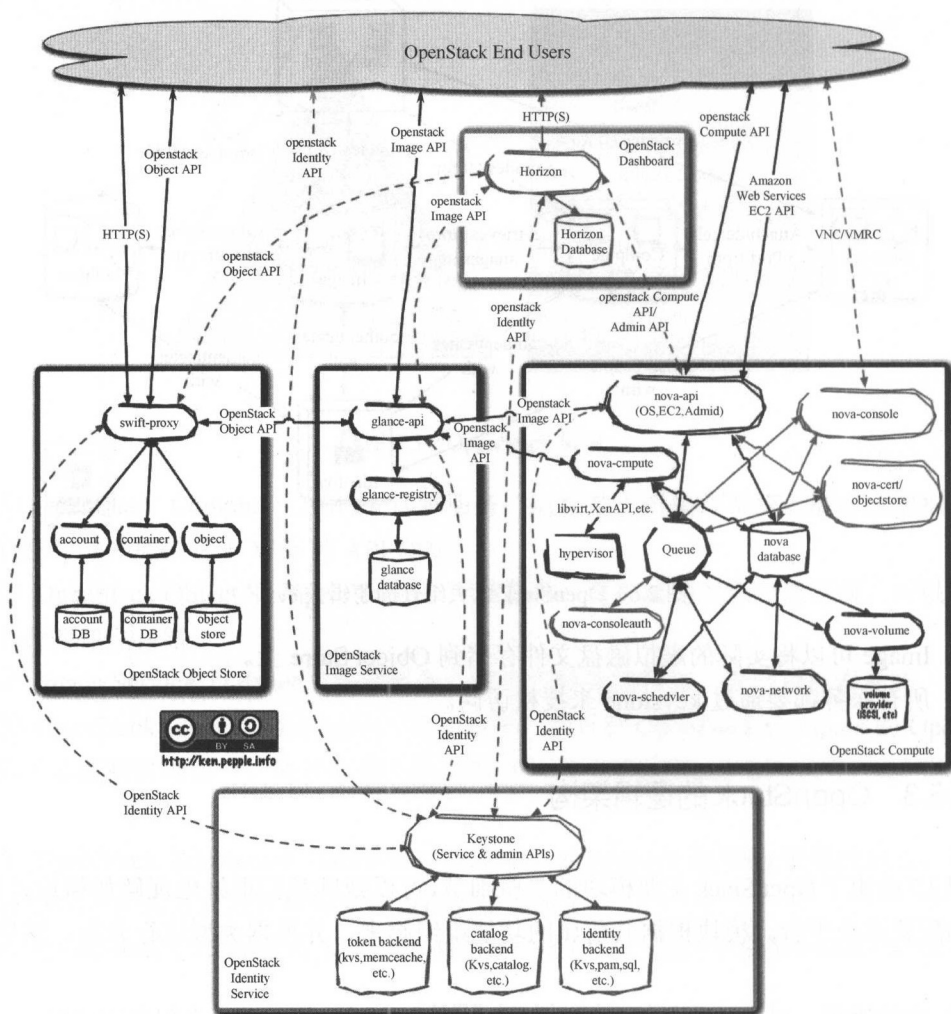


图 2.7 OpenStack 的逻辑架构

其各个组件的情况如下。

① nova-api 守护进程是 OpenStack Compute 的中心。它给所有 API 查询 (Compute API 或 EC2 API) 提供端点，部署活动 (比如运行实例)，以及实施一些策略 (绝大多数的配额检查)。

② nova-compute 进程主要是一个创建和终止虚拟机实例的守护进程。其过程相当复杂，但是基本原理很简单：从队列中接受行为，然后在更新数据库的状态时，通过一系列的命令执行。

③ nova-volume 负责管理映射到计算机实例的卷的创建、附加、取消和删除。这些卷可以来自很多提供商，比如 iSCSI 和 AOE。

④ nova network worker 守护进程类似于 nova-compute 和 nova-volume。它们从队列中接受网络任务，然后执行任务以操控网络，比如创建 bridging interfaces 或者改变 iptables rules。

⑤ Queue 提供中心 hub，为守护进程传递消息，当前用 RabbitMQ 实现，但是理论上可以是 Python amqp 支持的任何 AMPQ 消息队列。

⑥ nova database 存储云基础架构中的绝大多数编译时和运行时状态。这包括了可用的实例类型、在用的实例、可用的网络和项目。理论上，OpenStack Compute 能支持 SQL-Alchemy 支持的任何数据库，当前广泛使用 sqlite3（仅用于测试和开发工作）、MySQL 和 PostgreSQL。

⑦ OpenStack Glance 是一个单独的项目，它是一个 Compute 架构中可选的部分，分为 3 部分：glance-api、glance-registry 和 image store。其中，glance-api 接受 OpenStack image API 调用，glance-registry 负责存储和检索镜像的元数据，实际的 Image Blob 存储在 image store 中。Image Store 可以是多种不同的 ObjectStore，包括 OpenStack Object Storage（Swift）。

⑧ OpenStack Swift 是一个单独的项目，它采用分布式存储架构，能防止单点故障并支持横向扩展。它包括 4 部分：swift-proxy、account、container 和 object。swift-proxy 通过接收 OpenStack Object API 或者 HTTP 传入的请求，接受文件上传、修改元数据或容器创建。此外，它还将提供文件或容器清单到浏览器上。swift-proxy 可以使用一个可选的缓存（通常部署在 memcache 中）来提高性能。account 管理账户定义对象存储服务。container 管理一个映射的容器（即文件夹），提供对象存储服务。object 管理实际对象（例如文件）。

2.5.4 小结

OpenStack 很可能成为未来云计算平台的标准，只要遵循统一的标准，用户便可以随意将自己的应用部署到不同的云平台，而不需要对应用做任何修改。在未来统一的标准下，用户完全不用关心云服务提供商是用 OpenStack 构建的云还是其他平台构建云，只需要把应用部署到云即可，然后为使用的云资源付费。

2.6 软件定义网络

虚拟化技术是云计算发展的基础，云计算服务商以按需分配为原则，为客户提供具有高可用性、高扩展性的计算、存储和网络等 IT 资源。虚拟化技术将各种物理资源抽象为逻辑上的资源，隐藏了各种物理上的限制，为在更细粒度上对其进行管理和应用提供了可能性。近些年，计算的虚拟化技术（主要指 x86 平台的虚拟化）取得了长足的发展；相比较而言，尽管存储和网络的虚拟化也得到了诸多发展，但是还有很多问题亟待解决，在云计算环境中尤其如此。软件定义网络（Software Defined Network，SDN），是 Emulex 网络的一种新型网络创新架构，其核心技术 OpenFlow 通过将网络设备控制面与数据面分离开来，从而实现了网络流量的灵活控制。OpenFlow 和 SDN 尽管不是专门为网络虚拟化而生的，但是它们带来的标准化和灵活性却给网络虚拟化的发展带来了无限可能。

2.6.1 起源与发展

OpenFlow 起源于斯坦福大学的 Clean Slate 项目组。Clean Slate 项目的最终目的是要重新发明 Internet，改变设计已略显不合时宜，且难以进化发展现有的网络基础架构。在 2006 年，斯坦福的学生 Martin Casado 领导了一个关于网络安全与管理的项目 Ethane，该项目试图通过一个集中式的控制器，让网络管理员可以方便地定义基于网络流的安全控制策略，并将这些安全策略应用到各种网络设备中，从而实现对整个网络通信的安全控制。Martin 和他的导师 Nick McKeown 教授将传统网络设备的数据转发（Data Plane）和路由控制（Control Plane）两个功能模块相分离，通过集中式的控制器（Controller）以标准化的接口对各种网络设备进行管理和配置，那么这将为网络资源的设计、管理和使用提供更多的可能性，从而更容易推动网络的革新与发展。于是，他们便提出了 OpenFlow 的概念，并且 Nick McKeown 等人于 2008 年在 ACM SIGCOMM 发表了题为 OpenFlow: Enabling Innovation in Campus Networks 的论文，首次详细地介绍了 OpenFlow 的概念。该论文除了阐述 OpenFlow 的工作原理外，还列举了 OpenFlow 几大应用场景，包括：

- ① 校园网络中对实验性通讯协议的支持（如其标题所示）；
- ② 网络管理和访问控制；
- ③ 网络隔离和 VLAN；
- ④ 基于 WiFi 的移动网络；
- ⑤ 非 IP 网络；
- ⑥ 基于网络包的处理。

当然，目前关于 OpenFlow 的研究已经远远超出了这些领域。

基于 OpenFlow 为网络带来的可编程的特性，Nick 和他的团队进一步提出了 SDN（Software Defined Network）的概念。如果将网络中所有的网络设备视为被管理的资源，那么参考操作系统的原理，可以抽象出一个网络操作系统（Network OS）的概念，这个网络操作系统一方面抽象了底层网络设备的具体细节，同时还为上层应用提供了统一的管理视图和编程接口。这样，基于网络操作系统这个平台，用户可以开发各种应用程序，通过软件来定义逻辑上的网络拓扑，以满足对网络资源的不同需求，而无须关心底层网络的物理拓扑结构。

2.6.2 OpenFlow 标准和规范

自 2009 年年初发布第一个版本（v1.0）以来，OpenFlow 规范已经经历了 1.1、1.2、1.3 等版本。图 2.8 用来说明 OpenFlow 的原理和基本架构。其实，这张图还很好地表明了 OpenFlow Switch 规范所定义的范围，从图中可以看出，OpenFlow Switch 规范主要定义了 Switch 的功能模块以及其与 Controller 之间的通信信道等方面。Openflow 规范主要分为如

下 4 个部分。

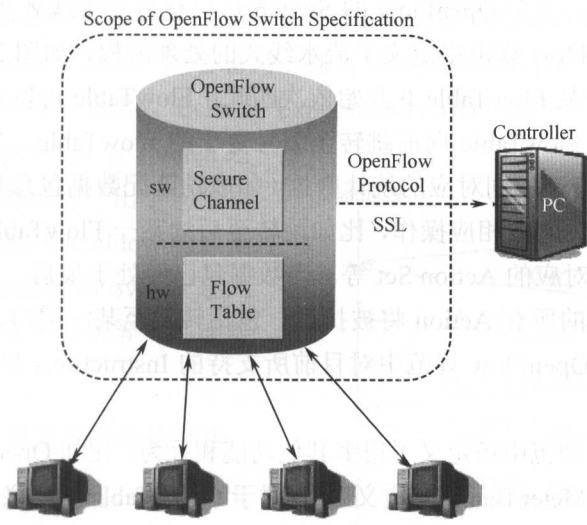


图 2.8 OpenFlow 的原理和基本架构

1. OpenFlow 的端口 (Port)

OpenFlow 规范将 Switch 上的端口分为 3 种类别。

- ① 物理端口，即设备上物理可见的端口。
- ② 逻辑端口，在物理端口基础上由 Switch 设备抽象出来的逻辑端口，如为 tunnel 或者聚合等功能而实现的逻辑端口。
- ③ OpenFlow 定义的端口。OpenFlow 目前总共定义了 ALL、CONTROLLER、TABLE、IN_PORT、ANY、LOCAL、NORMAL 和 FLOOD 8 种端口，其中后 3 种为非必需的端口，只在混合型的 OpenFlow Switch (OpenFlow-hybrid Switch，即同时支持传统网络协议栈和 OpenFlow 协议的 Switch 设备，相对于 OpenFlow-only Switch 而言) 中存在。

2. OpenFlow 的 FlowTable

OpenFlow 通过用户定义的或者预设的规则来匹配和处理网络包。一条 OpenFlow 的规则由匹配域 (Match Fields)、优先级 (Priority)、处理指令 (Instructions) 和统计数据 (如 Counters) 等字段组成，如图 2.9 所示。

Ingress Port	Ether Source	Ether Dst	Ether Type	Valn id	Vlan Priority	IP src	IP dst	IP proto	IP Tos bits	TCP/UDP Src Port	TCP/UDP Dst Port
--------------	--------------	-----------	------------	---------	---------------	--------	--------	----------	-------------	------------------	------------------

图 2.9 OpenFlow 规则

在一条规则中，可以根据网络包在 L2、L3 或者 L4 等网络报文头的任意字段进行匹配，比如以太网帧的源 MAC 地址，IP 包的协议类型和 IP 地址，或者 TCP/UDP 的端口号等。目前 OpenFlow 的规范中还规定了 Switch 设备厂商可以选择性地支持通配符进行匹配。

所有 OpenFlow 的规则都被组织在不同的 FlowTable 中，在同一个 FlowTable 中按规则的优先级进行匹配。一个 OpenFlow 的 Switch 可以包含一个或者多个 FlowTable，从 0 依次编号排列。OpenFlow 规范中定义了流水线式的处理流程，如图 2.10 所示。当数据包进入 Switch 后，必须从 FlowTable 0 开始依次匹配；FlowTable 可以按次序从小到大越级跳转，但不能从某一 FlowTable 向前跳转至编号更小的 FlowTable。当数据包成功匹配一条规则后，将首先更新该规则对应的统计数据（如成功匹配数据包总数目和总字节数等），然后根据规则中的指令进行相应操作，比如跳转至后续某一 FlowTable 继续处理，修改或者立即执行该数据包对应的 Action Set 等。当数据包已经处于最后一个 FlowTable 时，其对应的 Action Set 中的所有 Action 将被执行，包括转发至某一端口、修改数据包某一段、丢弃数据包等。OpenFlow 规范中对目前所支持的 Instructions 和 Actions 进行了完整详细的说明和定义。

另外，OpenFlow 规范中还定义了很多其他功能和行为，比如 OpenFlow 对于 QoS 的支持（即 MeterTable 和 Meter Bands 的定义等），对于 GroupTable 的定义，以及规则的超时处理等。

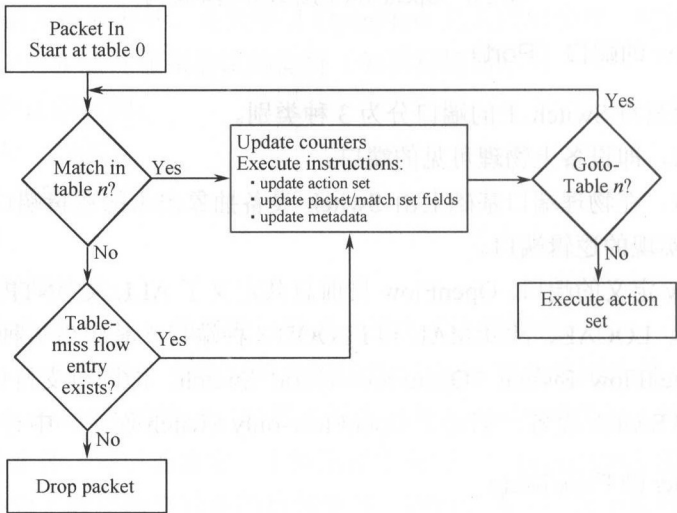


图 2.10 OpenFlow 中的处理流程

3. OpenFlow 的通信通道

OpenFlow 通信通道规范部分定义了一个 OpenFlow Switch 如何与 Controller 建立连接、通讯以及相关消息类型等的规范。OpenFlow 规范中定义了 3 种消息类型。

- ① Controller/Switch 消息，是指由 Controller 发起、Switch 接收并处理的消息，主要包括 Features、Configuration、Modify-State、Read-State、Packet-out、Barrier 和 Role-Request 等消息。这些消息主要由 Controller 用来对 Switch 进行状态查询和修改配置等操作。
- ② 异步（Asynchronous）消息，是由 Switch 发送给 Controller，用来通知 Switch 上发生的某些异步事件的消息，主要包括 Packet-in、Flow-Removed、Port-status 和 Error 等。例

如，当某一条规则因为超时而被删除时，Switch 将自动发送一条 Flow-Removed 消息通知 Controller，以方便 Controller 做出相应的操作，如重新设置相关规则等。

③ 对称（Symmetric）消息，顾名思义，都是双向对称的消息，主要用来建立连接、检测对方是否在线等，包括 Hello、Echo 和 Experimenter 3 种消息。

图 2.11 展示了 OpenFlow 和 Switch 之间一次典型的消息交换过程，出于安全和高可用性等方面的考虑，OpenFlow 的规范还规定了如何为 Controller 和 Switch 之间的信道加密，如何建立多连接等（主连接和辅助连接）。

4. OpenFlow 协议及相关数据结构

在 OpenFlow 规范的最后一部分，主要详细定义了各种 OpenFlow 消息的数据结构，包括 OpenFlow 消息的消息头等。这里就不一一赘述了，如需了解，可以参考 OpenFlow 源代码中 openflow.h 头文件中关于各种数据结构的定义。

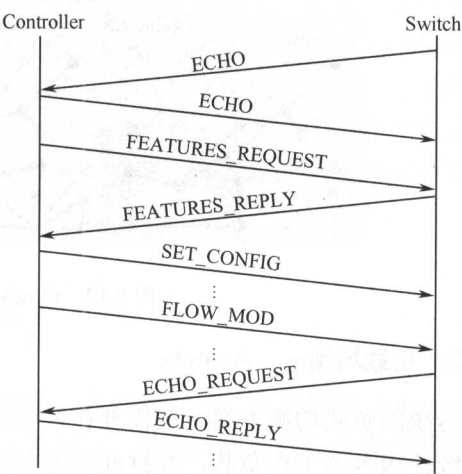


图 2.11 OpenFlow 和 Switch 之间消息交换过程

2.6.3 OpenFlow 的应用

随着 OpenFlow/SDN 概念的发展和推广，其研究和应用领域也得到了不断拓展。目前，关于 OpenFlow/SDN 的研究领域主要包括网络虚拟化、安全和访问控制、负载均衡、聚合网络和绿色节能等方面。另外，还有关于 OpenFlow 和传统网络设备交互及整合等方面的研究。下面将举几个典型的研究案例来展示 OpenFlow 的应用。

1. 网络虚拟化——FlowVisor

网络虚拟化的本质是要能够抽象底层网络的物理拓扑，能够在逻辑上对网络资源进行分片或者整合，从而满足各种应用对于网络的不同需求。为了达到网络分片的目的，FlowVisor 实现了一种特殊的 OpenFlow Controller，可以看成其他不同用户或应用的 Controllers 与网络设备之间的一层代理。因此，不同用户或应用可以使用自己的 Controllers 来定义不同的网络拓扑，同时 FlowVisor 又可以保证这些 Controllers 之间能够互相隔离而互不影响。图 2.12 展示了使用 FlowVisor 可以在同一个物理网络上定义出不同的逻辑拓扑。FlowVisor 不仅是一个典型的 OpenFlow 应用案例，同时还是一个很好的研究平台，目前已经有很多研究和应用都是基于 FlowVisor 做的。

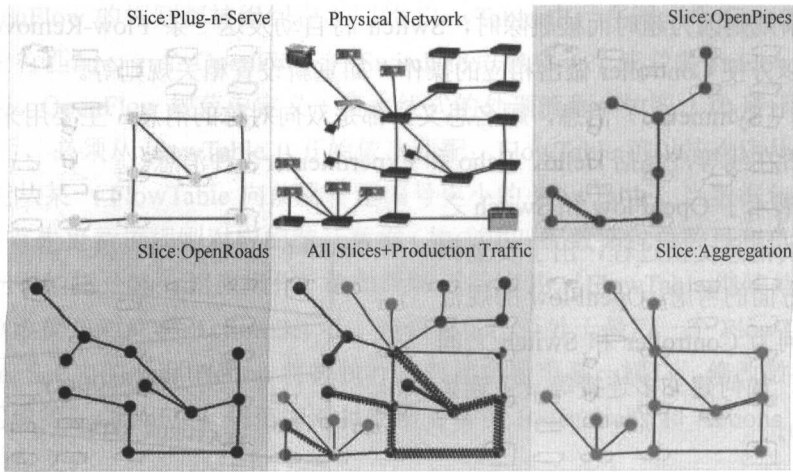


图 2.12 FlowVisor 定义的逻辑拓扑

2. 负载均衡——Aster*x

传统的负载均衡方案一般需要在服务器集群的入口处，通过一个网关或者路由器来监测、统计服务器工作负载，并据此动态分配用户请求到负载相对较轻的服务器上。既然网络中所有的网络设备都可以通过 OpenFlow 进行集中式的控制和管理，同时应用服务器的负载可以及时地反馈到 OpenFlow Controller 那里，那么 OpenFlow 就非常适合做负载均衡的工作。Aster*x 通过 Host Manager 和 Net Manager 来分别监测服务器和网络的工作负载，然后将这些信息反馈给 Flow Manager，这样 Flow Manager 就可以根据这些实时的负载信息，重新定义网络设备上的 OpenFlow 规则，从而将用户请求（即网络包）按照服务器的能力进行调整和分发（图 2.13）。

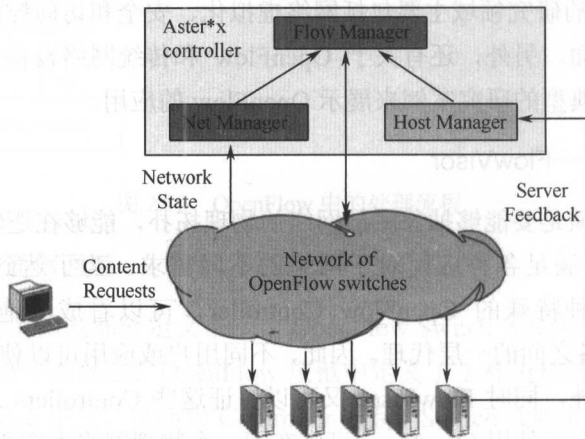


图 2.13 Aster*x 架构

2.7 虚拟机与容器

提到虚拟化技术大家肯定会想到虚拟机，也会想到 VMware、XEN、KVM、Hyper-V 这些产品。这种虚拟化我们可以简称为 VM (Virtual Machine) 虚拟化，就是以虚拟机为产物的虚拟化方案。还有一种虚拟化方案称为容器 (Container) 虚拟化方案，Container 是一种轻量级虚拟化方案，开销比 VM 虚拟化小，操作粒度也比 VM 虚拟化小。在云计算流行之前很多 IDC 的主机托管/租赁服务都是基于容器的方案。随着云计算对应用环境快速部署和对效率的要求不断提高，轻量级的容器虚拟化方案又重新获得青睐，当然相应的技术也经历了演进和革新。

2.7.1 VM 虚拟化与 Container 虚拟化

VM 虚拟化技术有 3 种：全虚拟化、半虚拟化、硬件虚拟化。全虚拟化由 Hypervisor 截获并翻译所有虚拟机特权指令（比如 VMware 的 BT）；半虚拟化通过修改虚拟机内核，将部分特权指令替换成与 Hypervisor（也称 VMM）通信（比如 XEN 的 para-virtualization）的指令；硬件虚拟化借助服务器硬件虚拟化功能，Hypervisor 不需要截获虚拟机特权指令，虚拟机也不需要修改内核（比如 Intel VT 和 AMD-V）。Hypervisor 负责服务器硬件资源管理，根据要求直接分配给不同虚拟机。Hypervisor 直接运行在服务器硬件上（半虚拟化/硬件虚拟化），也可以运行在一个操作系统上（全虚拟化模式）。

Container 虚拟化，又称操作系统级虚拟化，要求在一个操作系统实例里，将系统资源按照类型和需求分割给多个对象独立使用，对象之间保持隔离。系统资源通常指 CPU、内存、网卡、磁盘等。以 Linux Cgroup 为例，Cgroup 是 Linux 内核的一种文件系统，需要内核支持，和其他文件系统一样，Cgroup 在使用之前需要在 VFS 注册。用户可以直接使用 mount 命令挂载 Cgroup，通过 echo 命令修改 Cgroup 配置参数，跟环境变量一样，子进程可以继承父进程配置。Cgroup 提供 Linux 系统里进程的资源分配、资源使用情况统计。

VM 虚拟化与 Container 虚拟化各有优势，同时也存在着如下区别。

- ① 两者目标不同，VM 虚拟化的对象是虚拟机，把一台物理机虚拟成多台虚拟子机；Container 的操作对象是进程，为每个进程分配不同系统资源，进程与进程之间独立。
- ② VM 虚拟化组件可以直接运行在硬件之上，Container 只能运行在操作系统之上。
- ③ VM 虚拟组件负责管理物理机或虚拟子机的硬件资源；Container 环境中，硬件资源由操作系统自身负责管理。

2.7.2 Docker

Docker 是一个开源的应用容器引擎，其目标是实现轻量级的操作系统虚拟化解决方案。

Docker 的基础是 Linux 容器（LXC）等技术。在 LXC 的基础上 Docker 进行了进一步的封装，让用户不需要去关心容器的管理，使得操作更为简便。用户操作 Docker 的容器就像操作一个快速轻量级的虚拟机一样简单。图 2.14 比较了 Docker 和传统虚拟化方式的不同之处，可见容器在操作系统层面上实现虚拟化，直接复用本地主机的操作系统，而传统方式则在硬件层面实现。

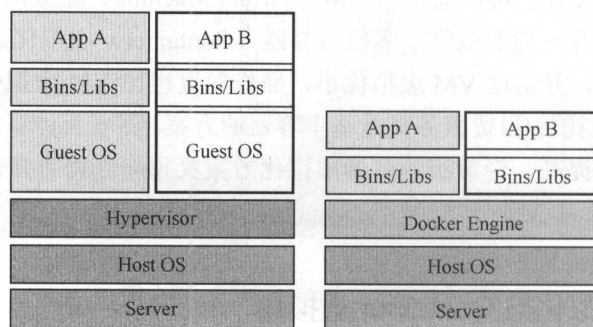


图 2.14 传统虚拟化方式和 Docker 的对比

作为一种新兴的虚拟化方式，Docker 跟传统的虚拟化方式相比具有众多的优势。首先，Docker 容器的启动可以在秒级实现，这相比传统的虚拟机方式要快得多。其次，Docker 对系统资源的利用率很高，一台主机上可以同时运行数千个 Docker 容器。容器除了运行其中应用外，基本不消耗额外的系统资源，使得应用的性能很高，同时系统的开销尽量小。传统虚拟机方式运行 10 个不同的应用就要建立 10 个虚拟机，而 Docker 只需要启动 10 个隔离的应用即可。具体来说，Docker 在如下几个方面具有较大的优势。

1. 更快速的交付和部署

对开发和运维人员来说，最希望的就是一次创建或配置，可以在任意地方正常运行。开发者可以使用一个标准的镜像来构建一套开发容器，开发完成之后，运维人员可以直接使用这个容器来部署代码。Docker 可以快速创建容器，快速迭代应用程序，并让整个过程全程可见，使团队中的其他成员更容易理解应用程序是如何创建和工作的。Docker 容器很轻很快，容器的启动时间是秒级的，大量地节约了开发、测试、部署的时间。

2. 更高效的虚拟化

Docker 容器的运行不需要额外的 Hypervisor 支持，它是内核级的虚拟化，因此可以实现更高的性能和效率。

3. 更轻松的迁移和扩展

Docker 容器几乎可以在任意的平台上运行，包括物理机、虚拟机、公有云、私有云、个人电脑、服务器等。这种兼容性可以让用户把一个应用程序从一个平台直接迁移到另外一个平台。

4. 更简单的管理

使用 Docker，只需要小小的修改，就可以替代以往大量的更新工作。所有的修改都以增量的方式被分发和更新，从而实现自动化并且高效的管理。

5. 对比传统虚拟机总结（表 2.3）

表 2.3 对比传统虚拟机总结

特性	容器	虚拟机
启动	秒级	分钟级
硬盘使用	一般为 MB	一般为 GB
性能	接近原生	弱于
系统支持量	单机支持上千个容器	一般几十个

2.8 练习题

- 1. 请谈谈对云计算的理解。
- 2. 简述云计算和大数据的关系。
- 3. 请简要概述云资源调度的几种策略。
- 4. 简述资源调度中的关键技术。
- 5. 为什么会出现多种调度策略？它们各有什么优缺点？
- 6. 简述 OpenStack 中关键组件的作用。
- 7. OpenFlow 的标准定义分为哪些部分？
- 8. 请阐述虚拟机虚拟化与容器虚拟化技术的区别，并探讨各自的优势。

参考文献

[1] Foster, Ian, Yong Zhao, Ioan Raicu, and Shiyong Lu. “Cloud computing and grid computing 360-degree compared.” In Grid Computing Environments Workshop, 2008. GCE'08, pp. 1-10. IEEE, 2008.

[2] Ahmad, Faraz, and T. N. Vijaykumar. “Joint optimization of idle and cooling power in data centers while maintaining response time.” ACM Sigplan Notices. Vol. 45. No. 3. ACM, 2010.

[3] Brunschwiler T,Smith B,Ruetsche E,et al. (2009),Toward zero-emission data centers through direct reuse of thermal energy. IBM J Res Dev 53(3):11.1–11.13.

[4] 张敏，陈云海．虚拟化技术在新一代云计算数据中心的应用研究[J]．广东通信技术，

2009(5).

- [5] Baliga, Jayant, et al. "Green cloud computing: Balancing energy in processing, storage, and transport." Proceedings of the IEEE 99.1 (2011): 149-167.
- [6] 潘春燕. 云计算实战: 把数据中心迁移到云环境[J]. 信息系统工程, 2009(2).
- [7] 刘鹏. 云计算 [M]. 北京: 电子工业出版社, 2010.
- [8] 陈康, 郑纬民. 云计算: 系统实例与研究现状 [J]. 软件学报, 2009, 20(5): 1337-1348.
- [9] 李知杰, 赵健飞. OpenStack 开源云计算平台[J]. 软件导刊, 2012, 11(12): 1-2.
- [10] 余侃. 云计算时代的数据中心建设与发展[J]. 信息通信, 2011(6): 1-3.
- [11] 李乔, 郑啸. 云计算研究现状综述[J]. 计算机科学, 2011, 38(4): 1-5.
- [12] Bein, Doina, Wolfgang Bein, and Shashi Phoha. "Efficient data centers, cloud computing in the future of distributed computing." Information Technology: New Generations (ITNG), 2010 Seventh International Conference on. IEEE, 2010.
- [13] Benini, Luca, et al. "Monitoring system activity for OS-directed dynamic power management." Proceedings of the 1998 international symposium on Low power electronics and design. ACM, 1998. pp.185-190.
- [14] 田文洪, 赵勇. 云计算——资源调度管理 [M]. 北京: 国防工业出版社, 2011.
- [15] 赵勇, 林辉, 沈寓实, 等. 大数据革命——理论、模式与技术创新 [M]. 北京: 电子工业出版社, 2014.
- [16] 蒋清野. 四大开源 IaaS 社区活跃度对比. CSDN, 2012.
- [17] McKeown, Nick, et al. "OpenFlow: enabling innovation in campus networks." ACM SIGCOMM Computer Communication Review 38.2 (2008): 69-74.
- [18] ONF Market Education Committee. "Software-Defined Networking: The new norm for networks." Palo Alto, California, USA: Open Networking Foundation (2012).
- [19] Sherwood, Rob, et al. "Flowvisor: A network virtualization layer." OpenFlow Switch Consortium, Tech. Rep (2009).
- [20] Handigol, Nikhil, et al. "Aster* x: Load-balancing as a network primitive." 9th GENI Engineering Conference (Plenary). 2010.
- [21] Heller, Brandon, et al. "ElasticTree: Saving Energy in Data Center Networks." NSDI. Vol. 10. 2010.

第 3 章

云计算先行者——Google 的三驾马车

3.1 Google 的三驾马车

云计算领域的领头羊和开篇之作，当属 Google 的三驾马车，Google 在 2003—2006 年间连续发表了三篇很有影响力的文章，分别是 2003 年 SOSP 的《The Google file system》，2004 年 OSDI 的《MapReduce: simplified data processing on large clusters》和 2006 年 OSDI 的《Bigtable: A distributed storage system for structured data》三篇论文分别介绍了 Google 分布式系统中所用到的 GFS、MapReduce 和 BigTable 的详细设计。GFS 是一个分布式文件系统，MapReduce 是一种用于大规模数据集的并行运算的编程模型，而 BigTable 则是一个非关系型的分布式数据存储系统。这三驾马车极大地推动了分布式系统和云计算的发展。Google 之所以能在当今互联网业取得巨大的成功，很大程度上是因为创立和应用了 GFS + MapReduce + BigTable 的底层架构。

3.1.1 GFS——一个可扩展的分布式文件系统

GFS (Google File System) 是 Google 自己研发的一个适用于大规模分布式数据处理相关应用的、可扩展的分布式文件系统。Google 之所以要研发这样一个分布式文件系统，是为了满足自己的业务需求，Google 的主要业务是搜索引擎，面对的是海量数据的处理，在此需求面前，传统的大型服务器太过昂贵，还有扩展性差等缺点。而普通硬件设备成本低廉，但可靠性较差。而 GFS 正是基于普通的不算昂贵的可靠性不高的硬件设备，实现了容

错的设计，并且为大量客户端提供了极高的聚合处理性能。

1. GFS 产生背景和简介

随着互联网的飞速发展以及 Google 自身的发展，原有的文件系统越来越不能满足 Google 的实际需求了，为了满足迅速增长的数据处理需求，Google 按照自己业务的实际需求设计了一套适用于大规模分布式数据处理的分布式文件系统——GFS。GFS 正好与 Google 的存储要求相匹配，并满足 Google 的服务产生和处理数据应用的要求，以及 Google 的海量数据的要求。因此 GFS 出现以来，就迅速在 Google 内部广泛用做存储平台。Google 最大的集群通过上千个计算机的数千个硬盘，提供了数百 TB 的存储，并且这些数据被数百个客户端并行操作。

GFS 使用廉价的普通硬件设备构建分布式文件系统，将容错的任务交由文件系统来完成，利用软件的方法解决系统可靠性问题，这样可以使得存储的成本大幅下降。由于 GFS 中服务器数目众多，在 GFS 中服务器死机是经常发生的事情，如何在频繁的故障中确保数据存储的安全、保证提供不间断的数据存储服务是 GFS 最核心的问题。GFS 的精彩在于它采用了多种方法，从多个角度，使用不同的容错措施来确保整个系统的可靠性。

2. GFS 设计预期

GFS 作为 Google 设计用来满足自己的业务需求的文件系统，必然会有一些特定的关注点，以下是 GFS 在设计时候遵循的一些目标和需求。

① 系统由许多廉价的普通组件组成，组件失效是一种常态而非异常。系统必须持续监控自身的状态，它必须将组件失效作为一种常态，能够迅速地侦测、冗余并恢复失效的组件。

② 系统存储一定数量的大文件。预期会有几百万文件，文件的大小通常在 100MB 或者以上。数个 GB 大小的文件也普遍存在，并且要能够被有效地管理。系统也必须支持小文件，但是不需要针对小文件做专门的优化。

③ 系统的工作负载主要由两种读操作组成：大规模的流式读取和小规模的随机读取。大规模的流式读取通常一次读取数百 KB 的数据，更常见的是一次读取 1MB 甚至更多的数据。来自同一个客户机的连续操作通常是读取同一个文件中连续的一个区域。小规模的随机读取通常是在文件某个随机的位置读取几个 KB 数据。如果应用程序对性能非常关注，通常的做法是把小规模的随机读取操作合并并且排序，之后按顺序批量读取，这样就避免了在文件中移动读取位置。

④ 系统的工作负载还包括许多大规模的、顺序的、数据追加方式的写操作。一般情况下，每次写入的数据的大小和大规模读类似。数据一旦被写入后，文件就很少会被修改了。当然系统也支持小规模的随机位置写入操作，但是可能效率不是很高。

⑤ 系统必须高效、行为定义明确地实现多客户端并行追加数据到同一个文件里的语义。Google 的文件通常被用于“生产者—消费者”队列，或者其他多路文件合并操作。通常会有数百个生产者，每个生产者进程运行在一台机器上，同时对一个文件进行追加操作。

使用最小的同步开销来实现的原子的多路追加数据操作是必不可少的。文件可以在稍后读取，或者是消费者在追加操作的同时读取文件。

⑥ 高性能的稳定网络带宽远比低延迟重要。Google 的目标程序绝大部分要求能够高速率、大批量地处理数据，极少有程序对单一的读、写操作有严格的响应时间要求。

3. GFS 系统架构

GFS 的系统架构如图 3.1 所示。GFS 将整个系统的节点分为 3 类角色：客户端 (Client)、主服务器 (Master) 和 Chunk 服务器。客户端是 GFS 提供给应用程序的访问接口，它是一组专用接口，不遵守 POSIX 规范，以库文件的形式提供。应用程序直接调用这些库函数，并与该库链接在一起。主服务器是 GFS 的管理节点，在逻辑上只有一个，它保存系统的元数据，负责整个文件系统的管理，是 GFS 文件系统上的“大脑”。Chunk 服务器负责具体的存储工作。数据以文件的形式存储在 Chunk 服务器上，Chunk 服务器的个数可以有多个，它的数目直接决定了 GFS 的规模。GFS 将文件按照固定大小进行分块，默认是 64MB，每一块称为一个 Chunk (数据块)，每个 Chunk 都有一个对应的索引号 (Index)。

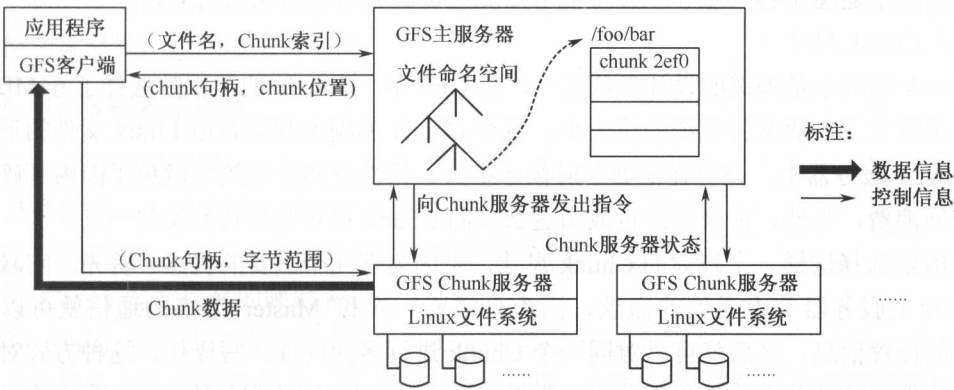


图 3.1 GFS 框架图

GFS 客户端在访问 GFS 时，首先访问主服务器节点，获取将要与之进行交互的 Chunk 服务器信息，然后直接访问这些 Chunk 服务器完成数据存取。GFS 的这种设计方法实现了控制流和数据流的分离，客户端与主服务器之间只有控制流，而无数据流，这样就极大地降低了主服务器的负载，使之不成为系统性能的一个瓶颈。客户端与 Chunk 服务器之间直接传输数据流，同时由于文件被分成多个 Chunk 进行分布式存储，客户端可以同时访问多个 Chunk 服务器，从而使得整个系统的 I/O 高度并行，系统整体性能得到提高。

下面将给出该架构中的一些特性的详细介绍。

(1) 单一主服务器节点

从 GFS 框架图可以看出，GFS 的主服务器节点在逻辑上只有一个，单一的主服务器节点的策略大大简化了 GFS 的设计。单一的主服务器节点可以通过全局的信息精确定位

Chunk 的位置以及进行复制决策。另外，应该尽可能减少对主服务器节点的读、写，以避免主服务器节点成为系统的瓶颈。因此客户端并不通过主服务器节点直接读写文件数据，而是向主服务器节点询问它应该联系的 Chunk 服务器。客户端将这些元数据信息缓存一段时间，后续的操作将直接和 Chunk 服务器进行数据读写操作。这也就是前面所说的客户端与主服务器之间只有控制流，而无数据流。

下面根据图 3.1 解释一次具体的读取流程。首先，客户端把文件名和程序指定的字节偏移，根据固定的 Chunk 大小，转换成文件的 Chunk 索引。然后，它把文件名和 Chunk 索引发送给主服务器节点。主服务器节点将相应的 Chunk 标识和副本的位置信息发还给客户端。客户端用文件名和 Chunk 索引作为键来缓存这些信息。

随后客户端发送请求到其中的一个副本处，考虑到效率和开销一般会选择最近的，请求信息包含了 Chunk 的标识和字节范围。在对这个 Chunk 的后续读取操作中，客户端不必再和主服务器节点通信，除非缓存的元数据信息过期或者文件被重新打开。实际上，客户端通常会在一次请求中查询多个 Chunk 信息，主服务器节点的回应也可能包含了紧跟着这些被请求的 Chunk 后面的 Chunk 的信息。在实际应用中，这些额外的信息在几乎没有任何代价的情况下避免了客户端和主服务器节点未来可能会发生的几次通信。

(2) Chunk 尺寸

Chunk 的大小是关键的设计参数之一。在 GFS 中，Google 的工程师选择了 64MB，这个尺寸远远大于一般文件系统的块大小。每个 Chunk 的副本都以普通 Linux 文件的形式保存在 Chunk 服务器上，只有在需要的时候才扩大。惰性空间分配策略避免了因内部碎片造成的空间浪费，当然，内部碎片也成为这么大的 Chunk 尺寸的最具争议的一点。

之所以选择这样一个较大的 Chunk 尺寸，是因为有几个重要的优点。首先，它减少了客户端和主服务器节点通信的需求，因为只需要一次和 Master 节点的通信就可以获取 Chunk 的位置信息，之后就可以对同一个 Chunk 进行多次的读、写操作。这种方式对降低文件系统的工作负载来说效果显著，因为 Google 的应用程序通常是连续读、写大文件。即使是进行小规模随机读取，采用较大的 Chunk 尺寸也带来了明显的好处，客户端可以轻松缓存一个数 TB 的工作数据集的所有 Chunk 位置信息。而且采用较大的 Chunk 尺寸使得客户端能够对一个块进行多次操作，这样就可以通过与 Chunk 服务器保持较长时间的 TCP 连接来减少网络负载。选用较大的 Chunk 尺寸减少了主服务器节点需要保存的元数据的数量。这就允许把元数据全部放在内存中，下面的内容将会分析元数据全部放在内存中带来的好处和缺陷。

(3) 元数据

主服务器（物理主服务器）存储 3 种主要类型的元数据，包括：文件和 Chunk 的命名空间、文件和 Chunk 的对应关系、每个 Chunk 副本的存放地点。所有的元数据都保存在主服务器的内存中。前两种类型的元数据（命名空间、文件和 Chunk 的对应关系）同时也会以记录变更日志的方式记录在操作系统的系统日志文件中，日志文件存储在本地磁盘上，同时日志会被复制到其他的远程主服务器上。采用保存变更日志的方式，能够简单可靠地

更新主服务器的状态，并且不用担心主服务器崩溃导致数据不一致的风险。主服务器不会持久保存 Chunk 位置信息。主服务器在启动时，或者有新的 Chunk 服务器加入时，向各个 Chunk 服务器轮询它们所存储的 Chunk 的信息。

前面说到了元数据是保存在内存中的，所以主服务器的操作速度非常快。并且，主服务器可以在后台简单而高效地周期性扫描自己保存的全部状态信息。这种周期性的状态扫描也用于实现 Chunk 垃圾收集、在 Chunk 服务器失效时重新复制数据、通过 Chunk 的迁移实现跨 Chunk 服务器的负载均衡以及磁盘使用状况统计等功能。

当然，将元数据全部保存在内存中的方法也有潜在问题：Chunk 的数量以及整个系统的承载能力都受限于主服务器所拥有的内存大小。但是在实际应用中，这并不是一个严重的问题。主服务器只需要不到 64 字节的元数据就能够管理一个 64MB 的 Chunk。由于大多数文件都包含多个 Chunk，因此绝大多数 Chunk 都是满的，除了文件的最后一个 Chunk 是部分填充的。同样地，每个文件在命名空间中的数据大小通常在 64 字节以下，因为保存的文件名是压缩过的。

即使需要支持更大的文件系统，为主服务器增加额外内存的费用其实是很少的，而通过增加这有限的费用，就能够把元数据全部保存在内存里，增强了系统的简洁性、可靠性、高性能和灵活性，显然这是可取的。

4. GFS 的容错机制

(1) 主服务器容错

具体来说，主服务器上保存了 GFS 文件系统的 3 种元数据：

- ① 命名空间，也就是整个文件系统的目录结构。
- ② Chunk 与文件名的映射表。
- ③ Chunk 副本的位置信息，每一个 Chunk 默认有 3 个副本。

首先就单个主服务器来说，对于前两种元数据，GFS 通过操作日志来提供容错功能。第三种元数据信息则直接保存在各个 Chunk 服务器上，当主服务器启动或 Chunk 服务器向主服务器注册时自动生成。因此当主服务器发生故障时，在磁盘数据保存完好的情况下，可以迅速恢复以上元数据。为了防止主服务器彻底死机的情况，GFS 还提供了主服务器远程的实时备份，主服务器逻辑上只有一个，但物理上可以有多台，其中一台扮演当前工作中的 GFS 主服务器的角色，这样在当前的 GFS 主服务器出现故障无法工作的时候，另外一台 GFS 主服务器可以迅速接替其工作。

(2) Chunk 服务器容错

GFS 采用副本的方式实现 Chunk 服务器的容错。每一个 Chunk 有多个存储副本（默认为三个），分布存储在不同的 Chunk 服务器上。副本的分布策略需要考虑多种因素，如网络的拓扑、机架的分布、磁盘的利用率等。对于每一个 Chunk，必须将所有的副本全部写入成功，才视为成功写入。在其后的过程中，如果相关的副本出现丢失或不可恢复等状况，主服务器会自动将该副本复制到其他 Chunk 服务器，从而确保副本保持一定的个数。尽管

一份数据需要存储 3 份，看起来磁盘空间的利用率并不高，但综合比较多种因素，采用副本无疑是最简单、最可靠、最有效，而且实现的难度也是最小的一种方法。

前面介绍过，GFS 中的每一个文件被划分成多个 Chunk，Chunk 的默认大小是 64MB。Chunk 服务器存储的是 Chunk 的副本，副本以文件的形式进行存储。而每一个 Chunk 以 Block 为单位进行划分，大小为 64KB，每一个 Block 对应一个 32bit 的校验和。当读取一个 Chunk 副本时，Chunk 服务器会将读取的数据和校验和进行比较，如果不匹配，就会返回错误，从而使客户端选择其他 Chunk 服务器上的副本。

5. 总结

GFS 展示了一个使用普通硬件支持大规模数据处理的系统的特质。GFS 成功地实现了 Google 对存储的需求，在 Google 内部，无论是作为研究和开发的存储平台，还是作为生产系统的数据处理平台，都得到了广泛的应用。它是 Google 持续创新和处理整个 Web 范围内的难题的一个重要工具。虽然一些设计要点都是针对 Google 的特殊需要定制的，但是还是有很多特性适用于类似规模和成本的数据处理任务，也就是说给其他公司的类似的需求提供了一个很好的解决思路，当前流行的开源系统 Hadoop 的文件系统 HDFS 也是基于 GFS 的思路的。GFS 的出现极大地推动了分布式系统的发展。如果读者想对 GFS 有更深入的了解，可以详细阅读论文《The Google File System》。

3.1.2 MapReduce——一种并行计算的编程模型

MapReduce 是一种编程模型，用于大规模数据集（大于 1TB）的并行运算。概念“Map”（映射）和“Reduce”（归约）及它们的主要思想，都是从函数式编程语言（Functional Language）里借用而来的，同时也包含了从矢量编程语言里借来的特性。MapReduce 极大地方便了编程人员在不会分布式并行编程的情况下，将自己的程序运行在分布式系统上。该编程模型由 Google 的 Jeffrey Dean 和 Sanjay Ghemawat 于 2004 年在论文里首次向外界公开。

1. MapReduce 产生背景和简介

在 Google 成立之初的几年里，包括原论文的作者在内的 Google 的很多程序员，为了处理海量的原始数据，已经实现了数以百计的、专用的计算方法。那些计算方法用来处理大量的原始数据，比如文档抓取、Web 请求日志等；也为了计算处理各种类型的衍生数据，比如倒排索引、Web 文档图结构的各种表示形式、每台主机上网络爬虫抓取的页面数量的汇总、每天被请求的最多的查询的集合等。大多数这样的数据处理运算在概念上很容易理解。然而由于输入的数据量巨大，因此要想在可接受的时间内完成运算，只有将这些计算分布在成百上千的主机上。如何处理并行计算、如何分发数据、如何处理错误等，这些问题综合在一起，需要大量的代码处理，因此也使得原本简单的运算变得难以处理。

为了解决上述复杂的问题，Google 的工程师设计了一个新的抽象模型，使用这个抽象

模型，使用者只表述想要执行的简单运算即可，而不必关心并行计算、容错、数据分布、负载均衡等复杂的细节，这些问题都被封装在了一个库里面。设计这个抽象模型的灵感来自 Lisp 和许多其他函数式语言的 Map 和 Reduce 的原语。实际情况是，Google 的应用中大数目的运算都包含这样的操作：在输入数据的逻辑记录上应用 Map 操作得出一个中间 key/value 对集合，然后在所有具有相同 key 值的 value 值上应用 Reduce 操作，从而达到合并中间的数据，得到一个想要的结果的目的。使用 MapReduce 模型，再结合用户实现的 Map 和 Reduce 函数，就可以非常容易地实现大规模并行化计算；而通过 MapReduce 模型自带的再次执行（Re-execution）功能，也提供了初级的容错实现方案。

2. MapReduce 编程模型

MapReduce 编程模型的原理是：利用一个输入 key/value 对集合来产生一个输出的 key/value 对集合。MapReduce 库的用户用两个函数表达这个计算：Map 和 Reduce。

用户自定义的 Map 函数接受一个输入的 key/value 对值，然后产生一个中间 key/value 对值的集合。MapReduce 库把所有具有相同中间 key 值 I 的中间 value 值集合在一起后传递给 reduce 函数。

用户自定义的 Reduce 函数接受一个中间 key 的值 I 和相关的一个 value 值的集合。Reduce 函数合并这些 value 值，形成一个较小的 value 值的集合。一般地，每次 Reduce 函数调用只产生 0 或 1 个输出 value 值。通常通过一个迭代器把中间 value 值提供给 Reduce 函数，这样就可以处理无法全部放入内存中的大量的 value 值的集合。

例如，计算一个大的文档集合中每个单词出现的次数，下面是伪代码段：

```
map(String key, String value):
    // key: document name
    // value: document contents
    for each word w in value:
        EmitIntermediate(w, "1");

reduce(String key, Iterator values):
    // key: a word
    // values: a list of counts
    int result = 0;
    for each v in values:
        result += ParseInt(v);
    Emit(AsString(result));
```

Map 函数输出文档中的每个词，以及这个词的出现次数（在这个简单的例子里就是 1）。Reduce 函数把 Map 函数产生的每一个特定的词的计数累加起来。

另外，用户编写代码，使用输入和输出文件的名字、可选的调节参数来完成一个符合 MapReduce 模型规范的对象，然后调用 MapReduce 函数，并把这个规范对象传递给它。用户的代码和 MapReduce 库链接在一起。

尽管在前面例子的伪代码中使用了以字符串表示的输入、输出值，但是在概念上，用户定义的 Map 和 Reduce 函数都有相关联的类型：

```
map(k1,v1) ->list(k2,v2)
```

```
reduce(k2,list(v2)) ->list(v2)
```

在上面的这两行定义中，输入的 key 和 value 值与输出的 key 和 value 值来自不同的域。而中间 key 和 value 值与输出 key 和 value 值来自相同的域。

除了上述词频统计的经典实例，还有一些有趣的简单例子，可以很容易地使用 MapReduce 模型来表示。

① 分布式的 Grep: Map 函数输出匹配某个模式的一行，Reduce 函数是一个恒等函数，即把中间数据复制到输出。

② 计算 URL 访问频率: Map 函数处理日志中 Web 页面请求的记录，然后输出 (URL, 1)。Reduce 函数把相同 URL 的 value 值都累加起来，产生 (URL, 记录总数) 结果。

③ 倒转网络链接图: Map 函数在源页面 (source) 中搜索所有的链接目标 (target) 并输出为 (target, source)。Reduce 函数把给定链接目标 (target) 的链接组合成一个列表，输出 (target, list (source))。

④ 每个主机的检索词向量: 检索词向量用一个 (词, 频率) 列表来概述出现在文档或文档集中的最重要的一些词。Map 函数为每一个输入文档输出 (主机名, 检索词向量)，其中主机名来自文档的 URL。Reduce 函数接收给定主机的所有文档的检索词向量，并把这些检索词向量加在一起，丢弃掉低频的检索词，输出一个最终的 (主机名, 检索词向量)。

⑤ 倒排索引: Map 函数分析每个文档输出一个 (词, 文档号) 的列表，Reduce 函数的输入是一个给定词的所有 (词, 文档号)，排序所有的文档号，输出 (词, list (文档号))。所有的输出集合形成一个简单的倒排索引，它以一种简单的算法跟踪词在文档中的位置。

⑥ 分布式排序: Map 函数从每个记录提取 key，输出 (key, record)。Reduce 函数不改变任何值。这个运算依赖分区机制和排序属性。

3. MapReduce 的具体实现

MapReduce 模型可以有多种不同的实现方式。如何正确选择取决于具体的环境。例如，一种实现方式适用于小型的共享内存方式的机器，另一种实现方式则适用于大型 NUMA (非一致存储访问) 架构的多处理器的主机，而有的实现方式更适合大型的网络连接集群。

这里给出一个 Google 内部广泛使用的运算环境，简而言之就是用以太网交换机连接、由普通 PC 组成的大型集群。环境里包括：

① x86 架构、运行 Linux 操作系统、双处理器、2G~4GB 内存的机器。

② 普通的网络硬件设备，每个机器的带宽为百兆或者千兆，但是远小于网络的平均带宽的一半。

- ③ 集群中包含成百上千的机器，因此，机器故障是常态。
- ④ 存储为廉价的内置 IDE 硬盘。一个内部分布式文件系统用来管理存储在这些磁盘上的数据。文件系统通过数据复制来在不可靠的硬件上保证数据的可靠性和有效性。
- ⑤ 用户提交工作（job）给调度系统。每个工作（job）都包含一系列的任务（task），调度系统将这些任务调度到集群中多台可用的机器上。
- ⑥ 下面将给出 MapReduce 在这个环境下的一个基本工作流程。

(1) 工作流程

通过将 Map 调用的输入数据自动分割为 M 个数据片段的集合，Map 调用被分布到多台机器上执行。输入的数据片段能够在不同的机器上并行处理。使用分区函数将 Map 调用产生的中间 key 值分成 R 个不同分区（例如， $\text{hash}(\text{key}) \bmod R$ ），Reduce 调用也被分布到多台机器上执行。分区数量（ R ）和分区函数由用户来指定。

图 3.2 是 MapReduce 实现中操作的全部流程。当用户调用 MapReduce 函数时，将发生下面一系列动作。

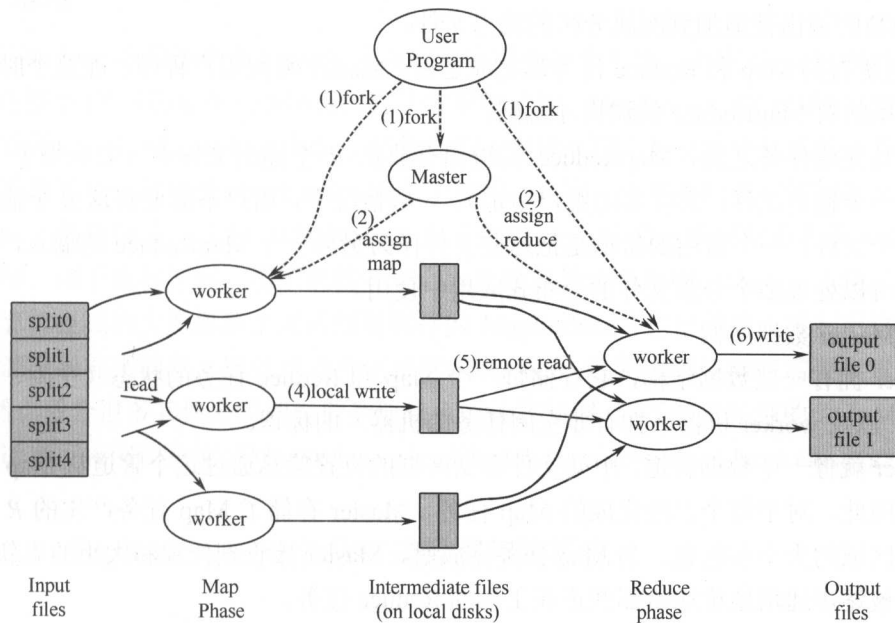


图 3.2 MapReduce 执行过程

① 用户程序首先调用 MapReduce 库将输入文件分成 M 个数据片段，每个数据片段的大小一般为 16M~64MB（可以通过可选的参数来控制每个数据片段的大小）。然后用户程序在机群中创建大量的程序副本。

② 这些程序副本中有一个特殊的程序——Master。副本中其他程序都是 worker 程序，由 Master 分配任务。有 M 个 Map 任务和 R 个 Reduce 任务将被分配，Master 将一个 Map 任务或 Reduce 任务分配给一个空闲的 worker。

③ 被分配了 Map 任务的 worker 程序读取相关的输入数据片段，从输入的数据片段中解析出 key/value 对，然后把 key/value 对传递给用户自定义的 Map 函数，由 Map 函数生成并输出的中间 key/value 对，并缓存在内存中。

④ 缓存中的 key/value 对通过分区函数分成 R 个区域，之后周期性地写入本地磁盘。缓存的 key/value 对在本地磁盘上的存储位置将被回传给 Master，由 Master 负责把这些存储位置再传送给 Reduce worker。

⑤ 当 Reduce worker 程序接收到 Master 程序发来的数据存储位置信息后，使用 RPC 从 Map worker 所在主机的磁盘上读取这些缓存数据。当 Reduce worker 读取了所有的中间数据后，通过对 key 进行排序后使得具有相同 key 值的数据聚合在一起。由于许多不同的 key 值会映射到相同的 Reduce 任务上，因此必须进行排序。如果中间数据太大无法在内存中完成排序，那么就要在外部进行排序。

⑥ Reduce worker 程序遍历排序后的中间数据，对于每一个唯一的中间 key 值，Reduce worker 程序将这个 key 值和它相关的中间 value 值的集合传递给用户自定义的 Reduce 函数。Reduce 函数的输出被追加到所属分区的输出文件。

⑦ 当所有的 Map 和 Reduce 任务都完成之后，Master 唤醒用户程序。在这个时候，在用户程序里的对 MapReduce 的调用才返回。

在成功完成任务之后，MapReduce 的输出存放在 R 个输出文件中（对应每个 Reduce 任务产生一个输出文件，文件名由用户指定）。一般情况下，用户不需要将这 R 个输出文件合并成一个文件，一个常用的做法是把这些文件作为另外一个 MapReduce 的输入，或者在另外一个可以处理多个分割文件的分布式应用中使用。

（2）Master 数据结构

Master 拥有一些数据结构，它存储每一个 Map 和 Reduce 任务的状态（空闲、工作中或完成），以及 worker 机器（拥有非空闲任务的机器）的标识。

Master 就像一个数据管道，中间文件存储区域的位置信息通过这个管道从 Map 传递到 Reduce。因此，对于每个已经完成的 Map 任务，Master 存储了 Map 任务产生的 R 个中间文件存储区域的大小和位置。当 Map 任务完成时，Master 接收到位置和大小的更新信息，这些信息被逐步递增地推送给那些正在工作的 Reduce 任务。

（3）存储位置

在很多的计算运行环境中，网络带宽是一个相当匮乏的资源。因此 MapReduce 通过尽量把输入数据（由 GFS 管理）存储在集群中机器的本地磁盘上来节省网络带宽。前面介绍 GFS 的时候讲到过，GFS 把每个文件按 64MB 一个块分隔，每个块保存在多台机器上，环境中就存放了多份副本（一般是 3 个副本）。MapReduce 的 Master 在调度 Map 任务时会考虑输入文件的位置信息，尽量将一个 Map 任务调度在包含相关输入数据副本的机器上执行；如果上述尝试失败了，Master 将试着在保存输入数据副本的机器附近的机器上执行 Map 任务（例如，分配到一个和包含输入数据的机器在一个 switch 里的 worker 机器上执行）。当在一个足够大的 cluster 集群上运行大型 MapReduce 操作的时候，大部分的输入数据都能从

本地机器读取，因此消耗的网络带宽非常少。

(4) 备用任务

影响一个 MapReduce 的总执行时间最通常的因素是“落伍者”：在运算过程中，如果有一台机器花了很长的时间才完成最后几个 Map 或 Reduce 任务，导致 MapReduce 操作总的执行时间超过预期。导致“落伍者”出现的原因非常多。比如：如果一个机器的硬盘出了问题，在读取的时候要经常地进行读取纠错操作，导致读取数据的速度大幅度降低。如果 cluster 的调度系统在这台机器上又调度了其他的任务，由于 CPU、内存、本地硬盘和网络带宽等竞争因素的存在，那么执行 MapReduce 代码的执行效率将会更加低下。

MapReduce 有一个通用的机制来减少“落伍者”出现的情况。当一个 MapReduce 操作接近完成的时候，Master 调度备用任务进程来执行剩下的、处于处理状态中的任务。无论是最初的执行进程，还是备用任务进程完成了任务，都把这个任务标记成为已经完成。采用这样的机制对于减少超大 MapReduce 操作的总处理时间有着显著的效果。

4. 总结

MapReduce 编程模型在 Google 内部成功应用于多个领域，而在 Google 将该模型公布以后，在整个 IT 界的分布式领域也得到了大量的应用。MapReduce 的成功可以归结为几个方面：首先，由于 MapReduce 封装了并行处理、容错处理、数据本地化优化、负载均衡等技术难点的细节，这使得 MapReduce 库易于使用。其次，大量不同类型的问题都可以通过 MapReduce 简单解决。比如，MapReduce 用于生成 Google 的网络搜索服务所需要的数据、用于排序、用于数据挖掘、用于机器学习，以及很多其他的系统；Google 实现了一个在数千台计算机组成的大型集群上灵活部署运行的 MapReduce，也就使得有效利用这些丰富的计算资源变得非常简单，因此也适合用来解决 Google 遇到的其他很多需要大量计算的问题，当然也适合很多其他公司的有着类似特点的计算问题。要了解更详细的关于 MapReduce 的内容读者可以阅读论文《MapReduce: simplified data processing on large clusters》。

3.1.3 BigTable——一个分布式数据存储系统

BigTable 是 Google 提出的一个管理结构化数据的分布式存储系统，可以理解为一个非关系的数据库。BigTable 的设计目的是可靠地处理 PB 级别的数据，并且能够部署到上千台机器上。Bigtable 实现了下面的几个目标：适用性广泛、可扩展、高性能和高可用性。

1. BigTable 产生背景和简介

Google 的很多项目，包括 Web 索引、Google Earth、Google Finance，对存储系统的要求差异非常大，无论是在数据规模（从 URL、网页到卫星图像）还是在响应速度上（从后端的批量处理到实时数据服务）。因此 Google 需要一个个灵活、高性能的解决方案来应对这些应用，Google 花了两年半的时间设计、实现并部署了一个用于管理结构化数据的分布

式的存储系统，也就是我们所说的 BigTable。BigTable 已经实现了下面几个目标：广泛的适用性、可扩展、高性能和高可用性。已经有超过 60 个 Google 的产品和项目在使用 BigTable，包括 Google Analytics、Google Finance、Orkut、Personalized Search、Writely 和 Google Earth。这些产品使用 Bigtable 完成迥异的工作负载需求，这些需求从面向吞吐量的批处理作业到对终端用户而言延时敏感的数据服务。它们使用的 BigTable 集群的配置也有很大的差异，从少数机器到成千上万台服务器，这些服务器里最多可存储几百 TB 的数据。

在很多方面，BigTable 和数据库很类似：它使用了很多数据库的实现策略。并行数据库和内存数据库已经具备可扩展性和高性能，但是 BigTable 提供了一个和这些系统完全不同的接口。BigTable 不支持完整的关系数据模型；与之相反，BigTable 为客户提供了简单的数据模型，利用这个模型，客户可以动态控制数据的布局和格式，用户也可以自己推测在底层存储中展示的数据的位置属性。数据用行和列的名字进行索引，名字可以是任意的字符串。虽然客户程序通常会在把各种结构化或半结构化的数据串行化到字符串里，BigTable 同样将数据视为未经解析的字符串。通过仔细选择数据的模式，客户可以控制数据的位置。最后，可以通过 BigTable 的模式参数动态地控制数据读或写。

2. BigTable 数据模型

BigTable 是一个稀疏的、分布式的、持久化存储的多维度排序 Map (key/value 对)。Map 由行关键字、列关键字以及时间戳索引，Map 中的每个 value 都是一个未经解析的字节数组：

(row:string, column:string,time:int64)->string

下面给出一个对 BigTable 设计影响很大的例子，这个例子使得 Google 的工程师做了很多设计决策。假设需要备份海量的网页及相关信息，这些数据可以用于很多不同的项目，称这个特殊的表为 Webtable。在 Webtable 里，使用 URL 作为行关键字，使用网页的各种属性 (aspect) 作为列名，网页的内容存在 contents:列中，并用获取该网页的时间戳作为标识，也就是可以根据时间存储网页的多个不同版本，如图 3.3 所示。

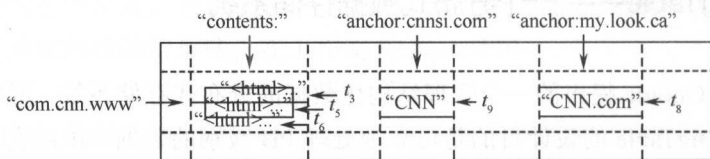


图 3.3 一个存储 Web 网页的例子的表的片段

图 3.3 是一个存储 Web 网页的例子的表的片段。行名是一个反向 URL。contents 列族包含的是网页的内容，anchor 列族包含引用该网页的锚链接文本。CNN 的主页被 Sports Illustrated 和 MY-look 的主页引用，因此该行包含了名为 “anchor:cnnsi.com” 和 “anchhor:my.look.ca” 的列。每个锚链接数据项只有一个版本；而 contents 列有 3 个版本，分别由时间戳 t_3 、 t_5 和 t_6 标识。下面将详细介绍行、列族以及时间戳的概念。

(1) 行

表中的行关键字是任意字符串。在单一行关键字下的每一个读或者写操作都是原子的（要么全做、要么全不做），这个设计决策能够使用户很容易地推测对同一个行进行并发更新操作时的系统行为。

BigTable 通过行关键字的字典顺序来维护数据。表中一定范围内的行被动态地分区。每个分区叫做一个“Tablet”，Tablet 是数据分布和负载均衡的单位。这样的好处是，读取一定范围内的少数行非常高效，并且往往只需要跟少数机器通信。用户可以通过选择他们的行关键字来开发这种特性，这样可以为他们的数据访问获得好的本地性。比如，假如 Google 要在关键字 `com.google.maps/index.html` 的索引下为 `maps.google.com/index.htm` 存储数据，那么把相同的域中的网页存储在连续的区域可以让一些主机和域名的分析更加有效。

(2) 列族

列关键字组成的集合叫做“列族”，列族构成了访问控制的基本单位。存放在同一列族下的所有数据通常都属于同一个类型。列族必须先创建，然后才能在列族中任何的列关键字下存放数据；列族创建后，其中的任何一个列关键字下都可以存放数据。BigTable 的设计意图是，一张表中不同列族的数目要小（最多几百个），并且列族在操作中很少改变。但是与此相反，一张表可以有无限多个列。

列关键字的命名语法为：列族：限定词。列族的名字必须是可打印的字符串，但是限定词可以是任意字符串。比如，Webtable 有一个列族 `language`，用来存放撰写网页的语言。在 `language` 列族中只使用一个列关键字，用来存放每个网页的语言标识 ID。Webtable 中另一个有用的列族是 `anchor`，这个列族的每一个列关键字代表单独一个锚链接，如图 3.3 所示。限定词是引用该网页的站点名，数据项内容是链接文本。

而访问控制、磁盘和内存的计数都是在列族层面进行的。在上面的 Webtable 的例子中，上述的控制权限能帮助使用者管理不同类型的应用：一些应用可以添加新的基本数据；一些可以读取基本数据并创建派生的列族；一些只允许浏览现存数据，甚至可能因为隐私的原因连浏览权限都没有。

(3) 时间戳

在 BigTable 中，每一个数据项都可以包含同一数据的不同版本，这些版本通过时间戳来索引。BigTable 时间戳是 64 位整型数。时间戳可由 BigTable 指定，这种情况下时间戳代表精确到毫秒的“实时”时间，或者该值由用户程序明确指定。需要避免冲突的程序必须自己生成一个唯一的时间戳。为了使最新的版本可以被先读到，数据项中不同版本是按照时间戳倒序排列的。

为了减轻多个版本数据的管理负担，对每一个列族提供两个设置参数，BigTable 通过这两个参数可以对废弃版本的数据进行自动垃圾收集。用户既可以指定只保存最后 n 个版本的数据，也可以只保存“足够新”的版本的数据（比如只保存最近 7 天写入的数据）。

在 Webtable 的例子中，Google 的工程师将存储在 `contents` 列中爬虫经过的页面的时间戳设置为这个版本的页面被实际爬过的时间。

3. BigTable 所需重要构件

BigTable 是建立在一些其他 Google 基础架构之上的,也就是说 BigTable 的实现需要其他一些构件的支持。BigTable 使用 GFS 存储日志和数据文件。BigTable 集群往往运行在一个共享的机器池中,池中的机器还会运行其他各种各样的分布式应用程序, BigTable 的进程经常要和其他应用的进程共享机器。BigTable 依赖集群管理系统在共享机器上调度作业、管理资源、处理机器的故障和监视机器的状态。

(1) SSTable 文件格式

BigTable 数据在内部使用 Google SSTable 文件格式存储。SSTable 提供一个从键(key)到值(value)的持久化、已排序、不可更改的映射(Map),这里的 key 和 value 都是任意的字节串。对 SSTable 提供了如下操作:查询与一个指定 key 值相关的 value,或者遍历指定 key 值范围内的所有键值对。从内部看, SSTable 是一连串的数据块(每个块的默认大小是 64KB,但是这个大小是可以配置的)。SSTable 使用块索引(通常存储在 SSTable 的最后)来定位数据块,在打开 SSTable 的时候,索引被加载到内存。一次查找可以通过一次磁盘搜索完成:首先执行二分查找在内存索引里找到合适数据块的位置,然后在从硬盘中读取合适的数据库块。也可以选择把整个 SSTable 都映射到内存中,这样的话就可以在不用访问硬盘的情况下执行查询搜索。

(2) Chubby 分布式锁服务

除了 GFS, BigTable 还依赖一个高可用的、持久化的分布式锁服务组件,叫做 Chubby, Chubby 也是 Google 云计算框架下的重要组成部分。一个 Chubby 服务包括了 5 个活动的副本,其中一个副本被选为 Master,并且积极处理请求。只有在大多数副本正常运行,并且彼此之间能够互相通信的情况下, Chubby 服务才是可用的。当有副本失效的时候,出现故障时 Chubby 使用 Paxos 算法保证副本的一致性。Chubby 提供了一个名字空间,里面包括了目录和小文件。每个目录或者文件可以当成一个锁使用,对文件的读写操作都是原子的。Chubby 客户程序库提供对 Chubby 文件的一致性缓存。每个 Chubby 客户程序都维护一个与 Chubby 服务的会话,如果客户程序不能在租约到期的时间内重新签订会话租约,那么这个会话就过期失效。当一个客户会话失效时,它拥有的锁和打开的文件句柄都失效了。Chubby 客户程序可以在 Chubby 文件和目录上注册回调函数,当文件或目录改变,或者会话过期时,回调函数会通知客户程序。

BigTable 使用 Chubby 完成以下各种任务:保证在任意时间最多只有一个活动的 Master;存储 BigTable 数据的引导程序的位置;发现 Tablet 服务器,以及在 Tablet 服务器失效时进行处理;存储 BigTable 的模式信息(每张表的列族信息);存储访问控制列表。如果 Chubby 长时间无法访问, BigTable 就会失效。

读者若想了解 Chubby 的详细设计,可以阅读 Google 的论文《The Chubby lock service for loosely-coupled distributed systems》。

4. BigTable 具体实现

BigTable 的实现有 3 个主要的组件：链接到每个客户程序的库、一个 Master 服务器和多个 Tablet 服务器。在一个集群中可以动态地添加或删除一个 Tablet 服务器来适应工作负载的变化。

Master 主要负责以下工作：为 Tablet 服务器分配 Tablets，检测新加入的或者过期失效的 Table 服务器、平衡 Tablet 服务器的负载，以及对 GFS 中的文件进行垃圾收集。除此之外，它还处理模式修改操作，例如建立表和列族。

每个 Tablet 服务器都管理一组 Tablet（通常每个 Tablet 服务器有数十个至上千个 Tablet）。Tablet 服务器处理它所加载的 Tablet 的读、写操作，以及分割增长导致过大的 Tablet。

和很多单主节点类型的分布式存储系统类似，客户数据都不经过 Master 服务器：客户程序直接和 Tablet 服务器通信来进行读、写操作。由于 BigTable 的客户程序不依赖 Master 服务器来获取 Tablet 的位置信息，大多数客户程序甚至完全不和 master 通信。因此，在实际应用中 Master 的负载很小。

一个 BigTable 集群存储了很多表，每个表包含了一组 Tablet，而每个 Tablet 包含了某个范围内的行的所有相关数据。初始状态下，每个表只有一个 Tablet 组成。随着表中数据的增长，它被自动分割成多个 Tablet，默认情况下每个 Tablet 的大小是 100M~200MB。

下面将介绍 BigTable 实现中一些具体的细节问题。

(1) Tablet 的位置信息

BigTable 使用一个 3 层的、类似于 B+树的结构存储 Tablet 的位置信息（图 3.4）。

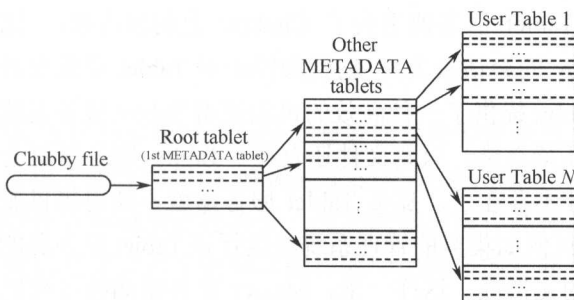


图 3.4 Tablet 位置层次结构

第一层是一个存储在 Chubby 中的文件，它包含了 Root Tablet 的位置信息。Root Tablet 在一个特殊的元数据（METADATA）表包含了里所有的 Tablet 的位置信息。每一个元数据 Tablet 包含了一组用户 Tablet 的位置信息。Root Tablet 实际上只是元数据表的第一个 Tablet，只不过对它的处理比较特殊，Root Tablet 永远不会被分割，这就保证了 Tablet 的位置层次不会超过 3 层。

元数据表将每个 Tablet 的位置信息存储在一个行关键字下，而这个行关键字是由 Tablet 所在的表的标识符和 Tablet 的最后一行编码而成的。每一个元数据行在内存中大约有 1KB

数据。在一个大小适中的、大小限制为 128MB 的元数据 Tablet 中，这样的三层结构位置信息模式足够寻址 234 个 Tablet（或者说在 128M 的元数据中可以存储 261 字节）。

客户程序库会缓存 Tablet 的位置信息。如果客户程序不知道一个 Tablet 的位置信息，或者发现它缓存的地址信息不正确，那么客户程序就递归移动到 Tablet 位置层次；如果客户端缓存是空的，那么寻址算法需要通过三次网络来回通信寻址，这其中包括了一次 Chubby 读操作。如果客户端缓存的地址信息过期了，那么寻址算法可能进行多达 6 次（3 次通信发现缓存过期，另外 3 次更新缓存数据）网络来回通信，因为过期缓存条目只有在没有查到数据的时候才能发现。尽管 Tablet 的位置信息是存放在内存里的，所以不用访问 GFS，但是，通常会通过预取 Tablet 地址来进一步减少访问开销：无论何时读取元数据表，都会为不止一个 Tablet 读取元数据。

在元数据表中还存储了次级信息，包括与 Tablet 有关的所有事件日志，这些信息有助于排除故障和性能分析。

（2）Tablet 分配

每个 Tablet 一次分配给一个 Tablet 服务器。Master 服务器记录活跃的 Tablet 服务器，当前 Tablet 到 Tablet 服务器的分配，以及哪些 Tablet 还没有被分配。当一个 Tablet 还没有被分配，并且刚好有一个 Tablet 服务器有足够的空闲空间装载该 Tablet 时，Master 服务器会给这个 Tablet 服务器发送一个装载请求，把 Tablet 分配给这个服务器。

BigTable 使用 Chubby 跟踪记录 Tablet 服务器的状态。当一个 Tablet 服务器启动时，它在 Chubby 的一个指定目录下建立一个有唯一性名字的文件，并且获取该文件的独占锁。Master 服务器实时监控着这个目录（服务器目录），因此 Master 服务器能够知道有新的 Tablet 服务器加入了。如果 Tablet 服务器丢失了 Chubby 上的独占锁，比如由于网络断开导致 Tablet 服务器和 Chubby 的会话丢失，它就会停止对 Tablet 提供服务。之所以能够采用这样的方法，是因为 Chubby 提供了一种高效的机制使得 Tablet 服务器能够在不增加网络负担的情况下知道它是否还持有锁。只要文件还存在，Tablet 服务器就会试图重新获得对该文件的独占锁；如果文件不存在了，那么 Tablet 服务器就不能再提供服务了，会自行退出。当 Tablet 服务器终止时（例如集群的管理系统将运行该 Tablet 服务器的主机从集群中移除），它会尝试释放它持有的文件锁，这样一来，Master 服务器就能尽快把 Tablet 分配到其他 Tablet 服务器。

Master 服务器负责检查一个 Tablet 服务器是否已经不再为它的 Tablet 提供服务了，并且要尽快重新分配它加载的 Tablet。Master 服务器通过轮询 Tablet 服务器文件锁的状态来检测何时 Tablet 服务器不再为 Tablet 提供服务。如果一个 Tablet 服务器向 Master 服务器报告它丢失了文件锁，或者 Master 服务器最近几次尝试和它通信都没有得到响应，Master 服务器就会尝试获取该 Tablet 服务器文件的独占锁；如果 Master 服务器成功获取了独占锁，那么就说明 Chubby 是正常运行的，而 Tablet 服务器要么是宕机了、要么是不能和 Chubby 通信了，在这种情况下 Master 服务器就删除该 Tablet 服务器在 Chubby 上的服务器文件以确保它不再给 Tablet 提供服务。一旦 Tablet 服务器在 Chubby 上的服务器文件被删除了，

Master 服务器就把之前分配给它的所有的 Tablet 放入未分配的 Tablet 集合中。为了确保 Bigtable 集群在 Master 服务器和 Chubby 之间网络出现故障的时候仍然可以使用，Master 服务器在它的 Chubby 会话过期后主动退出。如上所述，Master 服务器的故障不会改变现有 Tablet 在 Tablet 服务器上的分配状态。

(3) Tablet 服务

如图 3.5 所示，Tablet 的持久化状态信息保存在 GFS 上。更新操作提交到 REDO 日志中。在这些更新操作中，最近提交的那些存放在一个排序的缓存中，这个缓存称为 memtable；较早的更新存放在一系列 SSTable 中。为了恢复一个 Tablet，Tablet 服务器首先从 METADATA 表中读取它的元数据。Tablet 的元数据包含了组成这个 Tablet 的 SSTable 的列表，以及一系列的重做点（redo points），这些重做点指向可能含有该 Tablet 数据的已提交的日志记录。Tablet 服务器把 SSTable 的索引读进内存，之后通过重复重做点之后提交的更新来重建 memtable。

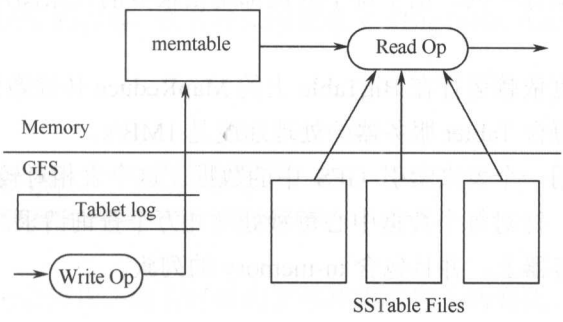


图 3.5 Tablet 表示

当对 Tablet 服务器进行写操作时，Tablet 服务器首先要检查这个操作格式是否正确、操作发起者是否有执行这个操作的权限。权限验证的方法是通过从一个 Chubby 文件里读取出来的具有写权限的操作者列表来进行验证。成功的修改操作会记录在提交日志里。可以采用批量提交方式来提高包含大量小的修改操作的应用程序的吞吐量。当一个写操作提交后，写的内容插入 memtable。

当对 Tablet 服务器进行读操作时，Tablet 服务器会做类似的完整性和权限检查。一个有效的读操作在一个由一系列 SSTable 和 memtable 合并的视图里执行。由于 SSTable 和 memtable 是按字典排序的数据结构，因此可以高效生成合并视图。

还有一点，当进行 Tablet 的合并和分割时，正在进行的读、写操作是能够继续进行的。

5. BigTable 实际应用

截至关于 BigTable 的论文发表前的 2006 年 8 月，Google 官方表示有 388 个非测试用的 BigTable 集群运行在各种各样的 Google 机器集群上，合计大约有 24500 个 Tablet 服务器。

以下介绍几个使用了 BigTable 的代表性的应用。

(1) Google Earth

Google 运转着一批为用户提供高分辨率地球表面卫星图像的服务，既可以通过基于 Web 的 Google Maps 接口 (maps.google.com)，也可以通过 Google Earth 可定制客户端软件访问。这些软件产品允许用户浏览地球表面：用户可以在许多不同的分辨率下平移、查看和注释这些卫星图像。这个系统使用一个表存储预处理数据，使用另外一组表存储用户数据。

预处理流水线使用一个表存储原始图像。在预处理过程中，图像被清除，然后被合并到最终的服务数据中。这个表包含了大约 70TB 的数据，因此需要从磁盘读取数据。图像已经被高效压缩过了，因此 BigTable 压缩在这里被禁用。

Imagery 表的每一行与一个单独的地理区块对应。行都有名称，以确保毗邻的区域存储在一起。Imagery 表中有一个记录每个区块的数据源的列族。这个列族包含了大量的列：基本上是一个原始数据图像一行。由于每个区块都是由很少的几张图片构成的，因此这个列族是很稀疏的。

预处理流水线高度依赖运行在 BigTable 上的 MapReduce 传输数据。在一些 MapReduce 作业中，整个系统中每台 Tablet 服务器的处理速度是 1MB/s。

这个服务系统使用一个表来索引 GFS 中的数据。这个表相对较小 (500GB)，但是这个表必须在低延迟下，针对每个数据中心每秒处理几万个查询请求。因此，这个表必须存储在上百个 Tablet 服务器上，并且包含 in-memory 的列族。

(2) 个性化查询

个性化查询是一个双向服务，这个服务记录用户的查询和点击，涉及各种 Google 的服务，比如 Web 查询、图像和新闻。用户可以浏览他们查询的历史，重复他们之前的查询和点击，用户也可以定制基于 Google 历史使用习惯模式的个性化查询结果。

个性化查询使用 BigTable 存储每个用户的数据。每个用户都有一个唯一的用户 ID，每个用户 ID 和一个列名绑定。一个单独的列族被用来存储各种类型的行为（例如有个列族可能是用来存储所有的 Web 查询的）。每个数据项都被用做 BigTable 的时间戳，记录了相应的用户行为发生的时间。个性化查询使用以 BigTable 为存储的 MapReduce 任务生成用户的数据图表。这些用户数据图表用来个性化当前的查询结果。

个性化查询的数据会复制到几个 BigTable 的集群上，这样就增强了数据可用性，同时减少了由客户端和 BigTable 集群间的“距离”造成的延时。个性化查询的开发团队最初建立了一个基于 BigTable 的、“客户侧”的复制机制为所有的复制节点提供一致性保障。现在的系统则使用了内建的复制子系统。

个性化查询存储系统的设计允许其他的团队在自己的列中加入新的用户数据，因此，很多 Google 服务使用个性化查询存储系统保存用户级的配置参数和设置。在多个团队之间分享数据的结果是产生了大量的列族。为了更好地支持数据共享，Google 加入了一个简单

的配额机制限制用户在共享表中使用的空间，配额也为使用个性化查询系统存储用户级信息的产品团体提供了隔离机制。

6. 总结

BigTable 作为 Google 的一个分布式的结构化数据存储系统，由于其优异的性能在面世后得到了 Google 的大量使用。Google 内部的团队对 BigTable 提供的高性能和高可用性很满意，随着时间的推移，他们可以根据自己的系统对资源的需求增加情况，通过简单地增加机器，扩展系统的承载能力。Google 的脚步没有停止，在论文发布后，Google 对 BigTable 加入了一些新的特性，比如支持二级索引，以及支持多 Master 节点、跨数据中心复制的 BigTable 的基础构件。Google 通过为 BigTable 设计自己的数据模型，使得他们的系统极具灵活性。另外，由于 Google 全面控制着 BigTable 的实现过程，以及 BigTable 使用到的其他的 Google 的基础构件，因此他们在系统出现瓶颈或效率低下的情况时能够快速解决问题。BigTable 中的优秀思想也由于其开源实现 HBase 的流行而造福了众多其他公司。要了解更详细的关于 BigTable 的内容，读者可以阅读论文《BigTable: A distributed storage system for structured data》。

3.2 Google 新“三驾马车”

技术的革新永远不会停止，随着时间的推移和需求的不断变化，Google 的三驾马车也显出了疲态。2009 年开始，网络巨头开始尝试使用新的技术取代 GFS 和 MapReduce。而 Google 也在自己的分布式系统中采用了新的技术以适应新的需求。因此也就出现了 Caffeine、Pregel 和 Dremel，也就是 Google 新三驾马车，这又是分布式系统领域的重大创新。

3.2.1 Caffeine——基于 Percolator 的搜索索引系统

Caffeine 是 Google 发布的新一代的整合能力更强、更快的搜索索引系统，Google 的老索引系统采用的是层级结构，顶层的内容比底层的内容更新要快，但是每一层的更新都需要 Google 扫描所有网络内容，然后再发现并排列新网页，费时又费力。Caffeine 颠覆了这种结构。它把更新任务分解成很多小块，不再需要对整个互联网信息链进行扫描，只需要持续不断关注每个小块，随时对其内容索引即可。这个新检索系统是全球搜索方法四年以来最大的变化。它现在每秒可以往 Google 索引库中添加上千的网页。

Caffeine 是基于 Percolator 的，而 Percolator 是一个由谷歌推出的、在海量数据（PB 级）上实现增量计算的平台，下面就给大家介绍 Percolator。

1. Percolator 背景与简介

数据收集和存储的速度正在惊人地发展,对 Google 而言,数以万计的服务器中存储的 PB 级数据,以及每天在服务器中处理的数以亿计的图片文件,都对其系统架构提出了新的挑战。重新设计系统架构以此优化搜索引擎的增量处理能力已是 Google 当务之急。

在此背景下,Percolator 诞生了,谷歌在 2010 年公开表示,将内容索引系统从 MapReduce 上迁移至 Percolator 上。目前该系统已经在 2010 年交付于 Google 的搜索索引内容,并按照系统需求有步骤地进行增加。按照 Google 官方给出的解释是,在 Google 的应用场景中,其作用在于将搜索的网页引入索引,在抓取文件的同时,Google 的主系统能够同时进行数据处理,系统可平均减少 50% 以上的延迟时间,甚至号称 Percolator 取代 MapReduce 之后,Google 的索引更新速度提升了 100 倍。

在搜索引擎系统中,文档被抓取后需要更新 Web 索引,新的文档会持续到达,这就意味着包含大量已存在索引的存储库需要不断变化。现实中有很多这样的数据处理任务,都是因为一些很小的、独立的变化导致一个大型仓库的转变。这种类型的任务的性能往往受制于已存在设施的容量。数据库能够很好地处理这种任务,但是它不适用于如此大规模的数据:Google 的索引系统存储了十几个 PB 的数据,并且每天在几千台机器上处理数十亿次更新。MapReduce 和其他批处理系统是为了大型批处理任务的效率而量身定制的,并不适合单独地处理小的更新。所以 Google 创建了 Percolator,一个在大型数据集上增量处理更新的系统,并且已经部署上线用于构建 Google 的 Web 搜索索引。通过将基于批处理的索引系统替换为 Percolator,Google 每天处理文档的数量相同,而搜索结果的年龄(从发布到被搜索的时间间隔)却减少了 50%。

Percolator 由 Google 的工程师在论文《Large-scale Incremental Processing Using Distributed Transactions and Notifications》中首次公开。

2. Percolator 设计思想

在 Web 索引系统中,系统开始会抓取互联网上的每一个页面,处理它们,同时在索引上维护一系列的不变量。比如,如果在多个 URL 下抓取到了相同的内容,只需要将 PageRank 最高的 URL 添加到索引中。每个外部链接也会被反向处理,让其锚文本附加到链接指向的页面上(链接中的锚文本往往能比较准确地评估其指向页面的内容)。链接反向处理还要考虑复制品(内容相同的多个页面):在必要的情况下指向一个复制品的链接应该被指向最高 PageRank 的页面(这样能增强最高 PageRank 的页面的评估)。

上述任务能够被表达为一系列的 MapReduce 操作:一个用于页面聚类分析,一个用于链接反向处理等。在 MapReduce 任务中维护不变量很简单,因为它是组织型计算(计算是按照一定逻辑和安排执行的,该并行的地方并行,该有序的地方有序),限制了计算的并行;所有文档都是按照步骤依次完成一个个阶段的处理。比如,当索引系统正附加锚文本到当前最高 PageRank 的 URL 时,我们不需要担心它的 PageRank 会并发改变:之前的 MapReduce 步骤已经完成了 PageRank 的计算,确定了它的 PageRank。

现在考虑在仅仅重新抓取了一小部分文档时该如何处理。对于 MapReduce 来说, 仅仅对新抓取的页面执行作业是不够的, 比如新来页面和已存在的老页面之间可能会有链接关系。MapReduce 必须在整个库之上再次运行, 换句话说, 运行 MapReduce 的对象既包括新页面也包括所有老页面。如果提供足够的计算资源, 加上 MapReduce 的可扩展性, 这个途径确实是可行的, 而且事实上, 在 Percolator 问世前, Google 的索引制造一直都按这种方式。然而, 对整个库再处理的做法丢掉了之前的工作成果, 延迟随着整个库的增长而成比例增长, 而不是这一次更新的量。

索引系统理论上可以将数据存储在一个数据库管理系统里, 这样就可以轻易实现只为单独的文档执行更新, 而且可以使用事务来维护不变量。然而, 当今的数据库管理系统不能处理数量如此庞大的数据: Google 的索引系统使用几千台机器存储了 10PB 的数据。像 BigTable 这样的分布式存储系统可以扩展到 Google 需要的容量, 但是在面对高并发更新时不能很好地帮助开发者维护不变量。

在 Google 工程师的眼里, 理想的处理系统是为增量处理优化定制的; 它应该允许他们维护一个非常大型的文档库, 并且当每一个新文档被抓取时高效率地更新。它可以高并发地处理很多小的更新, 而且要为并发更新维护不变量。Percolator 正是他们在此需求下所设计出来的一个出色的增量计算平台。

3. Percolator 功能与特性

Percolator 作为一个特殊的增量处理系统, 提供在 PB 级别存储库中随机访问的能力。随机访问允许使用者单独地处理文档, 避免全局扫描。为了达到高吞吐量, 它允许大量机器上的很多线程并发地对存储库执行更新, 所以 Percolator 为开发者提供了遵循 ACID 的事务机制, 目前是通过快照隔离语义来实现的。

为了解决并发问题, 增量系统的开发者需要持续跟踪增量计算的状态。Percolator 提供观察者来帮助实现此任务: 每当一个用户指定的列发生变化, 系统就调用一段代码逻辑。Percolator 应用的结构其实就是一系列的观察者; 每个观察者完成一个任务并通过对 Table 进行写操作来为“下游”的观察者创建更多的工作。一个外部的处理会将初始数据写入 Table, 以触发链路中的第一个观察者。

Percolator 是为增量处理量身定制的, 而且 Google 并不打算用它来代替已存在的大多数数据处理任务的解决方案, 因为如果任务结果不能被分解为小而多的更新 (比如文件排序), MapReduce 仍然是最好的选择。另外, 一致性很强的场景下才需要使用 Percolator: 否则 BigTable 就足够了。最后计算量也应该非常庞大: 计算量很小, 不需要用到 MapReduce 或 BigTable 的情况下, 一般的数据库管理系统就足够了。

在 Google, Percolator 的主要应用是实时构建 Web 搜索索引。索引系统使用 Percolator 之后, Google 能在文件被抓取时就单独地处理它。这几乎减少了 100 倍的平均文档处理延迟, 而且搜索结果中文档的平均年龄也降低了 50% (除了索引构建耗时, 搜索结果的年龄还包含文档从改变到被抓取之间的时间)。此系统也被用来将页面渲染为图片: Percolator

跟踪 Web 页面和它们依赖的资源之间的关系，所以当任何依赖的资源改变时页面也能够被再处理。

4. Percolator 创新之处

Percolator 得到巨大的速度提升并不仅仅是它的分布式事务和通知机制非常优秀，本质上的原因是它改变了索引构建系统的架构类型。比如对于 C2C 和 B2C 的购物平台，线上用来开展促销和秒杀活动的系统，要处理很大的数据量；而另一个系统，用于分析某个垂直行业的交易记录得出 BI 报表（比如 IT 数码行业各店铺的各类商品价格走势），也要处理很大的数据量。但是两者的架构类型却有本质上的不同。前者产生一笔交易的 input 数据很小，但是却要在已存在的海量数据中做多次查询（比如在支付阶段要在庞大数量的账户里查询买家账户和卖家账户，判断余额等），得到额外的 input，结合、计算之后产生的输出再插入茫茫数据。而后者，所有的输入已经确定，就是某段时间内的交易记录，不需要所谓的“已存在库”，更不需要在“已存在库”中查询额外的信息，只需要将交易记录执行广义上两个阶段的 MapReduce 处理，Map 阶段对原始数据进行解析，得出结构化的、细粒度的满足分析需要的数据，Reduce 阶段将有联系的数据汇聚到一起，按照一个公式执行计算，得到结果，保存入库。两种架构类型可以分别简称为增量系统、批处理系统。

那 Web 索引构建属于哪种系统呢？在以前，肯定属于批处理系统，因为网页已经被爬虫抓取过来了，存在磁盘上了，今天将其制造成索引，明天投入线上使用，供人查询。那个时候 Google 的核心矛盾不是索引更新得有多及时，而是用户输入搜索关键词之后响应有多快、返回的内容有多合适，一个新网页等个两三天才投入上线供人查询也完全没问题。

随着时间的推进，响应用户搜索这个环节已经不是核心矛盾了，已经做到非常优秀了。Google 开始不满足这种 T+1 的索引更新速度了，希望本质上提高时效性。而希望提升时效性，无非就是两个方向，一是继续批处理系统的模式，想办法提升性能，做各种优化；二是做成一个在线服务系统（增量系统）。很显然，Percolator 选择了第二条路，Google 能做出这样的架构决策，再追求极致地弥补它的缺陷，令人钦佩。

选择本质上的改革，变成一个在线服务系统，增量处理新来数据，提升了多倍的速度，Percolator 的强大的性能正是得益于系统定位和架构观念上的改变。从 MapReduce 到 Percolator，与其说是架构的变化，倒不如说是因为 Google 生态圈里的发展而导致索引生成系统的定位发生了变化。

3.2.2 Pregel——高效的分布式图计算的计算框架

Pregel 是 Google 的一个用于分布式图计算的计算框架，主要用于图遍历（BFS）、最短路径（SSSP）、PageRank 计算等。共享内存的图计算运行库有很多，但是对于 Google 来说，一台机器早已经放不下需要计算的数据了，所以需要分布式的计算环境。在 Pregel 之前，用 MapReduce 可以完成相关任务，但是效率很低；也可以利用已有的并行图算法库

Parallel BGL 或者 CGMgraph, 但是这两者又没有容错, 所以 Google 就自己开发了这个新的计算框架。Pregel 由 Google 在 2010 年的 SIGMOD 上发表的论文《Pregel: A System for Large-Scale Graph Processing》首次提出。

1. Pregel 产生背景和简介

Internet 的流行使得 Web Graph 成为人们争相分析和研究的热门对象。Web 2.0 更是激发了人们对社交网络的关注。其他的一些大型图对象(如交通路线图、新闻文章的相似性、疾病暴发路径、发表的科学研究文章中的引用关系等), 也已经被研究了数十年了。经常被用到的一些算法, 包括最短路径算法、不同种类的聚类算法、各种 PageRank 算法变种, 还有其他许多具有实际价值的图计算问题, 比如最小切割、连通分支。

如今随着计算机和计算机网络的发展, 所需要处理的图对象也越来越巨大。对大型图对象进行高效的处理, 是非常具有挑战性的。图算法常常表现出比较差的内存访问局部性, 针对单个顶点的处理工作过少, 以及计算过程中伴随着的并行度的改变等问题。分布式架构的采用更是加剧了位置问题, 并且增加了在计算过程中机器发生故障的概率。尽管大型图对象无处不在, 而且其在商业上的重要性不言而喻, 但是在 Google 的 Pregel 之前, 基本不存在一种在大规模分布式环境下, 可以基于各种图表示方法来实现任意图算法的、可扩展的通用系统。

一般来讲, 要实现一种处理大规模图对象的算法通常意味着要在以下几点中做出选择。

① 为特定的图应用定制相应的分布式实现, 这样的话在面对新的图算法或者图表示方式时, 就需要做大量的重复实现, 通用性差。

② 基于现有的分布式计算平台, 而这种情况下, 通常它们并不适于作图处理。比如 MapReduce 就是一个对许多大规模计算问题都非常合适的计算框架。有时它也被用来对大规模图对象进行挖掘, 但是通常在性能和易用性上都不是最优的。尽管这种对数据处理的基本模式经过扩展, 已经可以使用方便的聚合以及类 SQL 的查询方式, 但这些扩展对于图算法这种更适合用消息传递模型的问题来说, 通常并不理想。

③ 使用单机的图算法库, 如 BGL, LEAD, NetworkX, JDSL, Stanford GraphBase, FGL 等, 但这样对可以解决的问题的规模有很大的限制。

④ 使用已有的并行图计算系统。Parallel BGL 和 CGMgraph 这些库实现了很多并行图算法, 但是并没有解决对大规模分布式系统中来说非常重要的容错等一些问题。

而以上的这些选择都或多或少存在一些局限性。为了解决大型图的分布式计算问题(Google 最初需要解决的是 PageRank 计算问题), Google 搭建了一套可扩展、有容错机制的平台, 该平台提供了一套非常灵活的 API, 可以描述各种各样的图计算, 而它实现的就是一个通用的图计算框架——Pregel。

Pregel 作为一种适于处理大规模图的计算模型, 程序使用一系列的迭代过程来表达, 在每一次迭代中, 每个顶点会接收来自上一次迭代的信息, 并发送信息给其他顶点, 同时可能修改其自身状态及以它为顶点的出边的状态, 或改变整个图的拓扑结构。这种以顶点

为中心的策略非常灵活，足以用来表达一大类的算法。该模型的设计目标就是可以高效、可扩展和容错地在由上千台机器组成的集群中得以实现。此外它的隐式的同步性使得程序本身很容易理解，分布式相关的细节被隐藏在一组抽象出来的 API 下，这样展现给人们的就是一个具有丰富表现力、易于编程的大规模图处理框架。

2. Pregel 基本思想

对 Pregel 计算系统的灵感来自 Valiant 提出的 BSP (Bulk Synchronous Parallel) 模型。Pregel 的计算过程由一系列被称为超级步 (superstep) 的迭代 (iteration) 组成。在每一个超级步中，计算框架都会针对每个顶点调用用户自定义的函数，这个过程是并行的 (同一时刻可能有多个顶点被调用)。该函数描述的是一个顶点 V 在一个超级步 S 中需要执行的操作。该函数可以读取前一个超级步 ($S-1$) 中发送给 V 的消息，并发送消息给其他顶点，这些消息将会在下一个超级步 ($S+1$) 中被接收，并且在此过程中修改顶点 V 及其出边的状态。消息通常沿着顶点的出边发送，但一个消息可能会被发送到任意已知 ID 的顶点上去。

这种以顶点为中心的策略很容易让人联想起 MapReduce，因为这二者都让用户只需要关注其本地的执行逻辑，每条记录的处理都是独立的 (相互之间不需要通信)，系统将这些行为组合起来就可以完成大规模数据的处理。根据设计，这种计算模型非常适合分布式的实现：它没有将任何检测执行顺序的机制暴露在单个超级步中，所有的通信都仅限于 S 到 ($S+1$) 之间。

模型的同步性使得在实现算法时很容易理解程序的语义，并且使得 Pregel 程序天生免疫异步系统中经常出现的死锁以及临界资源竞争。理论上，Pregel 程序的性能即使在与足够并行化的异步系统的对比中都有一定的竞争力。因为通常情况下图计算的应用中顶点的数量要远远大于机器的数量，所以必须要平衡各机器之间的负载，这样各个超级步间的同步就不会增加过多的延迟，因为负载平衡会引入大量的通信开销，就使得超级步间的同步开销不那么明显了。

3. Pregel 计算模型

在 Pregel 计算模型中，输入是一个有向图，该有向图的每一个顶点都有一个相应的由 String 描述的顶点标识符。每一个顶点都有一个与之对应的可修改的用户自定义值。每一条有向边都和其源顶点关联，并且也拥有一个可修改的用户自定义值，并同时记录了其目标顶点的标识符。

一个典型的 Pregel 计算过程如下：读取输入初始化该图，当图被初始化好后，运行一系列的超级步直到整个计算结束，这些超级步之间通过一些全局的同步点分隔，输出结果结束计算。

在每个超级步中，顶点的计算都是并行的，每个顶点执行相同的用于表达给定算法逻辑的用户自定义函数。每个顶点可以修改其自身及其出边的状态，接收前一个超级步 ($S-1$) 中发送给它的消息，并发送消息给其他顶点 (这些消息将会在下一个超级步中被接收)，甚至是修改整个图的拓扑结构。需要注意的是，边在这种计算模式中并不是核心对象，没

有相应的计算运行在边上。

算法是否能够结束取决于是否所有的顶点都已经“vote”标识其自身已经达到“halt”状态了。在第 0 个超级步，所有顶点都处于 active 状态，所有的 active 顶点都会参与所有对应超级步中的计算。顶点通过将其自身的 status 设置成“halt”来表示它已经不再 active。这就表示该顶点没有进一步的计算需要执行，除非再次被外部触发，而 Pregel 框架将不会在接下来的超级步中执行该顶点，除非该顶点收到其他顶点传送的消息。如果顶点接收到消息被唤醒进入 active 状态，那么在随后的计算中该顶点必须显式地 deactive。整个计算在所有顶点都达到“inactive”状态，并且没有 message 在传送的时候宣告结束。这种简单的状态机如图 3.6 所示。

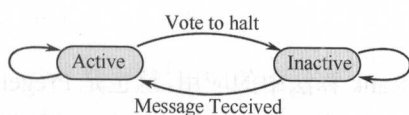


图 3.6 顶点状态机

整个 Pregel 程序的输出是所有顶点输出的集合。通常都是一个跟输入同构的有向图，但是这并不是必要的，因为顶点和边可以在计算的过程中进行添加和删除。比如一个聚类算法，就有可能是从一个大图中生成的非连通顶点组成的小集合，一个对图的挖掘算法就可能仅仅输出了从图中挖掘出来的聚合数据等。

图 3.7 通过一个简单的例子来说明这些基本概念：给定一个强连通图，图中每个顶点都包含一个值，它会将最大值传播到每个顶点。在每个超级步中，顶点会从接收到的消息中选出一个最大值，并将这个值传送给其所有的相邻顶点。当某个超级步中已经没有顶点更新其值，那么算法就宣告结束（图中虚线表示信息，阴影顶点已被投票终止）。

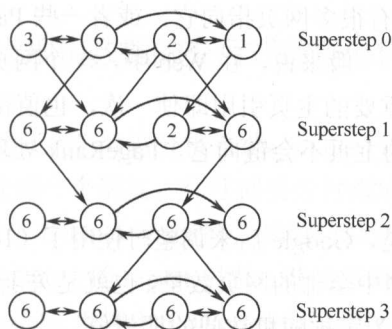


图 3.7 最大值示例

Pregel 选择了一种纯消息传递模型，忽略远程数据读取和其他共享内存的方式，有两个原因。第一，消息传递有足够的表达能力，没必要使用远程读取。Google 的工程师表示没有发现哪种算法是消息传递所不能表达的。第二，出于性能的考虑。在一个集群环境中，

从远程机器上读取一个值会有很高的延迟，这是很难避免的，而 Pregel 的消息传递模式通过异步和批量的方式传递消息，可以缓解这种远程读取的延迟。

图算法其实也可以被写成是一系列的链式 MapReduce 调用。之所以 Pregel 选择了另外一种不同的模式的原因在于可用性和性能。Pregel 将顶点和边保存在执行计算的那台机器上，而仅仅利用网络来传输信息。而 MapReduce 本质上是函数式的，所以将图算法用链式 MapReduce 来实现就需要将整个图的状态从一个阶段传输到另外一个阶段，这样就带来了许多的通信开销和随之而来的序列化和反序列化的开销。另外，这一连串的 MapReduce 作业各执行阶段需要的协同工作也增加了编程复杂度，而在 Pregel 中通过引入超级步避免了这样的情况。

4. Pregel 的应用

下面给出 Pregel 在 PageRank 算法中的应用，这也是 Pregel 最初被 Google 使用的地方。首先来简要介绍一下 PageRank 算法：将文献检索中的引用理论用到 Web 中，引用网页的链接数一定程度上反映了该网页的重要性和质量。PageRank 发展了这种思想，网页间的链接是不平等的。PageRank 定义如下：我们假设 $T_1 \dots T_n$ 指向网页 A 。参数 d 是制动因子，取值在 0, 1 之间。通常 d 等于 0.85。网页 A 的 PageRank 值由下式给出：

$$PR(A) = (1-d) + d(PR(T_1)/C(T_1) + \dots + PR(T_n)/C(T_n))$$

PageRank 或 $PR(A)$ 可以用简单的迭代算法计算，计算过程是收敛的，随着迭代次数的增加，各网页的 PageRank 值趋于平稳，可以从如下角度进行理解。

① 假设网上冲浪是随机的，不断点击链接，从不返回，最终厌倦了而另外随机选一个网页重新开始冲浪。随机访问一个网页的可能性就是它的 PageRank 值。制动因子 d 是随机访问一个网页厌倦了的可能性。对单个网页或一组网页，一个重要的变量会被加入制动因子 d 中。这允许个人可以故意地误导系统，以得到较高的 PageRank 值。

② 一般来说，一个网页有很多网页指向它，或者一些 PageRank 值高的网页指向它，则这个网页很重要。同样地，一般来说，在 Web 中，一个网页被很多网页引用，那么这个网页值得一看。一个网页被重要的主页引用即使一次，也值得一看。如果一个网页的质量不高，或者是死链接，重要的主页不会链向它。PageRank 处理了这两方面因素，并通过网络链接递归地传递。

关于该公式需要说明的是，Google 后来调整时使用了 $(1-d)/N$ ，公式的其他部分未做任何修改，这里的 N 是互联网中全部的网页数量，也就是关于 Pregel 的论文中使用的公式，据说这样做使得 PageRank 变为了被随机访问的期望值。

PageRank 算法的 Pregel 实现如下所示。PageRankVertex 继承自 Vertex 类。顶点 value 类型是 double，用来保存 PageRank 中间值，消息类型也是 double，用来传输 PageRank 分数，边的 value 类型是 void，因为不需要存储任何信息。假设在第 0 个超级步时，图中各顶点的 value 值被初始化为 $1/\text{NumVertices}()$ 。在前 30 个超级步中，每个顶点都会沿着它的出边发送它的 PageRank 值除以出边数后的结果值。从第 1 个超级步开始，每个顶点会收到

达的消息中的值加到 sum 值中,同时将它的 PageRank 值设为 $0.15 / \text{NumVertices}() + 0.85 \times \text{sum}$ 。到了第 30 个超级步后,就没有需要发送的消息了,同时所有的顶点 VoteToHalt。在实际中,PageRank 算法需要一直运行直到收敛,可以使用 aggregators 来检查是否满足收敛条件。

```
class PageRankVertex
{
public Vertex<double, void, double> {
public:
virtual void Compute(MessageIterator<msg> msgs) {
    if (superstep() >= 1) {
        double sum = 0;
        for (; !msgs->Done(); msgs->Next())
            sum += msgs->Value();
        *MutableValue() =
            0.15 / NumVertices() + 0.85 * sum;
    }
    if (superstep() < 30) {
        const int64 n = GetOutEdgeIterator().size();
        SendMessageToAllNeighbors(GetValue() / n);
    } else {
        VoteToHalt();
    }
}
};
```

3.2.3 Dremel——大规模数据的交互式数据分析系统

Dremel 是 Google 的“交互式”数据分析系统,可以组建成规模的集群,处理 PB 级别的数据。以往 MapReduce 处理一个数据,往往需要分钟级的时间,而作为 MapReduce 的提出者,Google 进一步开发了 Dremel 将处理时间缩短到秒级,作为 MapReduce 的有力补充。Dremel 作为 Google BigQuery 的 report 引擎,获得了很大的成功。而正是由于其大大优于 MapReduce 的性能,Apache 也推出了 Dremel 的开源实现 Drill。

1. 产生背景和简介

大规模分析型数据处理,在互联网公司乃至整个行业中都已经越来越广泛。目前已经可以用廉价的存储来收集和保存海量的企业核心业务数据,但是,更重要的是,必须懂得如何让分析师和工程师便捷地利用这些数据。在数据探测、监控、在线用户支持、快速原型设计、数据管道调试以及其他任务中,交互的响应时间是一个至关重要的系统设计考虑

因素。

执行大规模交互式数据分析对并行计算能力要求很高，例如，假设使用普通的硬盘，如果希望在 1 秒内读取 1TB 的压缩数据，那么就需要成千上万块硬盘。类似地，CPU 密集的查询操作也需要运行在成千上万个核上，并在数秒内完成。在 Google 公司里，大量的并行计算是使用普通 PC 组成的共享集群完成的。一个集群通常会部署大量的“共享资源、产生不同负载、需要不同硬件参数”的分布式应用。对于一个分布式应用而言，某个工作任务可能会比其他任务花费更多的时间，或者可能由于故障，或者被集群管理系统停止而永远不能完成。因此，处理好异常和故障是实现快速执行和容错的重要因素。

互联网和科学计算中的数据经常是没有关联的，因此，在这些领域，一个灵活的数据模型是十分必要的。在编程语言中使用的数据结构、分布式系统之间交换的消息、结构化文档等，都可以用嵌套方式来很自然地描述。嵌套数据模型已经成为 Google 处理大部分结构化数据的基础，不光是 Google，嵌套数据模型在其他公司也渐渐得到了广泛的应用。Dremel 正是这样一个基于嵌套数据模型的大规模交互式数据分析系统。

2. 交互式查询处理的必要性

我们首先来看一个场景，这个场景说明了交互式查询处理的必要性，以及它在数据管理生态系统上怎么定位。假设一个 Google 的员工 Alice，诞生了一个新奇的灵感，她想从 Web 网页中提取新类型的 signals。她运行一个 MapReduce 任务，分析输入数据然后产生这种 signals 的数据集合，在分布式文件系统上存储这数十亿条记录。为了分析她实验的结果，她启动 Dremel 然后执行几个交互式命令：

```
DEFINE TABLE t AS /path/to/data/*  
SELECT TOP(signal1, 100), COUNT(*) FROM t
```

她的命令只需几秒就执行完毕。她也运行了几个其他的查询来证实她的算法是正确的。她发现 signal1 中有非预期的情况，于是写了个 FlumeJava 程序执行了一个更加复杂的分析式计算。一旦这个问题解决了，她建立一个管道，持续地处理输入数据。然后她编写了一些 SQL 查询来跨维度地聚合管道的输出结果，然后将它们添加到一个交互式的 dashboard，其他工程师能非常快速地定位和查询它。

上述案例要求在查询处理器和其他数据管理工具之间互相操作，因此需要构建一个互相协作的数据管理组件来实现交互式查询处理。

3. Dremel 的组成要素

为了构建一个互相协作的数据管理组件，首先需要有一个通用存储层。GFS 是 Google 公司中广泛使用的分布式存储层。GFS 使用冗余复制来保护数据不受硬盘故障影响，即使出现“掉队者”也能达到快速响应时间。对原位数据管理来说，一个高性能的存储层是非常重要的。它允许访问数据时不消耗太多时间在加载阶段。这个要求也导致数据库在分析型数据处理中不太被使用。使用 GFS 的另外一个好处是，在文件系统中能使用标准工具便捷地操作数据，比如，迁移到另外的集群，改变访问权限，或者基于文件名定义一个数据

子集。

构建一个互相协作的数据管理组件还需要一个共享的存储格式。列状存储已经证明了它适用于扁平的关系型数据，但是要使它适用于 Google 则需要适配到一个嵌套数据模型。图 3.8 展示了主要的思想：一个嵌套字段比如 A.B.C，它的所有值被连续存储。因此，A.B.C 被读取时，不用读取 A.E、A.B.D 等。最终在 Dremel 里，Google 采用了一种嵌套列状存储。

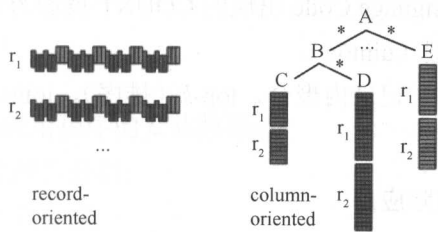


图 3.8 嵌套数据的行式存储和列式存储

4. Dremel 查询语言

Dremel 的查询语言基于 SQL，其实现是定制化设计的，可在列状嵌套存储上高效执行。定义语言严格上不是本文范畴，这里简单介绍一下它的特点。每个 SQL 语句以一个或多个嵌套表格和它们的 schema 作为输入，输出一个嵌套表格和它的 schema。下面描述了一个 query 例子，执行了投影（projection）、选择（selection）和记录内聚合等操作。例子中的查询执行在图 3.8 中的 $t = \{r_1, r_2\}$ 表格上。字段通过路径表达式来引用。查询最终根据某种规则产出一个嵌套结构的数据，不需要用户在 SQL 中指明构造规则。

```
SELECT DocId AS Id,
      COUNT(Name.Language.Code) WITHIN Name AS Cnt,
      Name.Url + ',' + Name.Language.Code AS Str
FROM t
WHERE REGEXP(Name.Url, '^http') AND DocId < 20;

Id: 10
Name
  Cnt: 2
  Language
    Str: 'http://A,en-us'
    Str: 'http://A,en'
Name
Cnt: 0

t1
message QueryResult {
  required int64 Id;
  repeated group Name {
    optional uint64 Cnt;
    repeated group Language {
      optional string Str;
    }
  }
}
```

为了解释查询做了什么，需要解释选择和投影两个操作。在选择操作中，可以将一个

嵌套记录看成一个树结构，树中每个节点的标签对应字段的名字。选择操作要做的，就是砍掉不满足指定条件的分支。因此，上述例子中，只有当 Name.Url 有值且满足正则 ‘^http’ 才被保留。下一步，在投影操作中，SELECT 子句中的每个标量表达式都会投影为一个值，此值的嵌套深度和表达式中重复字段最多的保持一致。所以，Str 值的嵌套深度与 Name.Language.Code 相同。COUNT 表达式部分用到了记录内聚合。每个 Name 子记录都会执行此聚合，将 Name.Language.Code 出现的 COUNT 投影为每个 Name 下的 Cnt 值，它是一个非负数的 64 位的整型（uint64）。

此语言支持嵌套子查询，记录内聚合，top-k（排序），joins（多表关联），用户自定义函数等。

5. Dremel 的特点和实际应用

前面介绍了关于 Dremel 的一些基本内容，总结起来它主要具有以下几个特点。

（1）Dremel 是一个大规模、稳定的系统

在一个 PB 级别的数据集上，将任务缩短到秒级，无疑需要大量的并发。Google 一向善于利用廉价机器组成强大的系统，但是，随着机器越来越多，出问题的概率也越来越大。如此大的集群规模，需要有足够的容错考虑，保证整个分析的速度不被集群中的个别不良节点所影响。

（2）Dremel 是 MapReduce 交互式查询能力不足的补充

和 MapReduce 一样，Dremel 也需要和数据运行在一起，将计算移动到数据上面，所以，它需要 GFS 这样的文件系统作为存储层。在设计之初，Dremel 并非是 MapReduce 的替代品，它只是可以执行非常快的分析，在使用的时候，常常用它来处理 MapReduce 的结果集或者用来建立分析原型。

（3）Dremel 的数据模型是嵌套的

互联网数据常常是非关系型的，这就要求 Dremel 必须有一个灵活的数据模型，这个数据模型对于获得高性能的交互式查询而言至关重要。因此，Dremel 采用了嵌套数据模型，这有点类似于 Json。嵌套数据模型相对于关系模型而言具有明显的优势。对于传统的关系模型而言，不可避免地存在大量 Join 操作，因此，在处理如此大规模的数据的时候，往往是有心无力的。而嵌套数据模型却可以在 PB 级别数据上一展身手。

（4）Dremel 中的数据是用列式存储的

当采用列式存储时，在分析的时候就可以只扫描需要的那部分数据，从而大大减少 CPU 和磁盘的访问量。同时，列式存储是压缩友好的，可以实现更高的压缩率，使得 CPU 和磁盘发挥最大的效能。对于关系型数据而言，在如何使用列式存储方面，整个行业都已经很有经验。但是，对于嵌套结构，Dremel 也通过巧妙的设计来实现列式存储，这是非常值得我们学习的。

(5) Dremel 结合了 Web 搜索和并行 DBMS 的技术

首先,它借鉴了 Web 搜索中的“查询树”的概念,将一个相对巨大、复杂的查询分割成较小、较简单的查询。大事化小,小事化了,能并发地在大量节点上运行。其次,和并行 DBMS 类似,Dremel 提供了一个类似 SQL 的接口,就像 Hive 和 Pig 那样。

Dremel 自从 2006 年开始就已经投入开发了,并且在 Google 公司已经有了几千用户。多种多样的 Dremel 实例被部署在 Google 公司里,每个实例拥有着数十至数千个节点。使用 Dremel 系统的例子包括:

- ① 分析网络文档;
- ② 追踪 Android 市场应用程序的安装数据;
- ③ Google 产品的崩溃报告分析;
- ④ Google Books 的 OCR 结果 ;
- ⑤ 垃圾邮件分析;
- ⑥ Google Maps 中的地图部件调试;
- ⑦ 管理中的 BigTable 实例的 Tablet 迁移;
- ⑧ Google 分布式构建系统中的测试结果分析;
- ⑨ 数万个硬盘的磁盘 I/O 统计信息;
- ⑩ Google 数据中心上运行的任务的资源监控;
- ⑪ Google 代码库的符号和依赖关系分析。

3.3 练习题

1. Google 的三驾马车分别都解决了什么问题?
2. 利用 MapReduce 实现词频统计。
3. Google 最新的大数据技术相比之前的有何改进。
4. Dremel 的存储格式有何特点?

参考文献

- [1] Ghemawat S, Gobioff H, Leung S T. The Google file system[C]//ACM SIGOPS Operating Systems Review. ACM, 2003, 37(5): 29-43.
- [2] 谷歌三大核心技术(一) The Google File System 中文版. <http://blog.csdn.net/hguisu/article/details/7244798>
- [3] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters[J]. Communications of the ACM, 2008, 51(1): 107-113.

- [4] Mapreduce 中文版. <http://blog.csdn.net/active1001/article/details/1675920>
- [5] Chang F, Dean J, Ghemawat S, et al. Bigtable: A distributed storage system for structured data[J]. ACM Transactions on Computer Systems (TOCS), 2008, 26(2): 4.
- [6] 谷歌三大核心技术 (三) Google BigTable 中文版. <http://blog.csdn.net/hguisu/article/details/7244991>
- [7] Google 后 Hadoop 时代的新“三驾马车”——Caffeine、Pregel、Dremel. <http://www.csdn.net/article/2012-08-20/2808870>
- [8] Peng D, Dabek F. Large-scale Incremental Processing Using Distributed Transactions and Notifications[C]//OSDI. 2010, 10: 1-15.
- [9] 经典论文翻译导读之《Large-scale Incremental Processing Using Distributed Transactions and Notifications》. <http://www.importnew.com/2896.html>
- [10] Malewicz G, Austern M H, Bik A J C, et al. Pregel: a system for large-scale graph processing[C]//Proceedings of the 2010 ACM SIGMOD International Conference on Management of data. ACM, 2010: 135-146.
- [11] Pregel: A System for Large-Scale Graph Processing(译). <http://duanple.blog.163.com/blog/static/70971767201281610126277/>
- [12] Melnik S, Gubarev A, Long J J, et al. Dremel: interactive analysis of web-scale datasets[J]. Proceedings of the VLDB Endowment, 2010, 3(1-2): 330-339.
- [13] 经典论文翻译导读之《Dremel: Interactive Analysis of WebScale Datasets》. <http://www.importnew.com/2617.html>
- [14] Google Dremel 原理——如何能 3 秒分析 1PB. <http://www.yankay.com/google-dremel-rationale/>

第4章

云存储系统

云存储不是一个设备，而是一种服务，具体来说，它是把数据存储和访问作为一种服务，并通过网络提供给用户。云计算是提供计算能力，相应地，云存储是提供存储能力。

云存储专注于向用户提供以网络为基础的在线存储服务，通过规模化来降低用户使用存储的成本。用户无须考虑存储容量、存储设备的类型、数据存储的位置以及数据完整性保护和容灾备份等烦琐的底层技术细节，按需付费就可以从云存储供应商那里获得近乎无限大的存储空间和企业级的服务质量。本章主要介绍云存储系统，从云存储的基础概念出发，介绍云存储涉及的关键技术，并对云存储系统按分类进行描述。

4.1 云存储的基本概念

云存储是在云计算概念上延伸和发展出来的一个新的概念，是指通过集群应用、网络技术或分布式文件系统等功能，将网络中大量各种不同类型的存储设备通过应用软件集合起来协同工作，共同对外提供数据存储和业务访问功能的一个系统。

4.1.1 云存储结构模型

随着宽带网络的发展、Web 2.0 技术的普及，很多云存储厂商在云计算所引发的浪潮中如雨后春笋般冒了出来，其中亚马逊的 AWS S3 是最具有代表性的。在中国，越来越

多的互联网公司也推出了他们的云存储服务。比如国内的百度云网盘、金山快盘、微博微盘、腾讯微云、360 云盘等，国外的 Dropbox，都有很大的用户量，其中国内的一些网盘更是为用户提供了多达 2TB 的免费存储空间。这些云存储服务的出现，也为我们的资料保存、分发、共享提供了极大的便利。

云存储实际是网络上所有的服务器和存储设备构成的集合体，其核心是用特定的应用软件来实现存储设备向存储服务功能的转变，为用户提供一定类型的数据存储和业务访问服务。与传统的存储设备相比，云存储不仅仅是一个硬件，而且是一个网络设备、存储设备、服务器、应用软件、公用访问接口、接入网、客户端程序等多个部分组成的复杂系统。各部分以存储设备为核心，通过应用软件来对外提供数据存储和业务访问服务。

为了解释云存储系统的结构模型，在这里借用互联网的结构模型来参考。相信大家对于局域网、广域网和互联网的一些概念都比较清楚，在常见的局域网系统中，我们为了更好地使用局域网，一般来讲，使用者需要非常清楚地知道网络中每一个软硬件的型号和配置，比如采用什么型号交换机，有多少个端口，采用了什么路由器和防火墙，分别是如何设置的。系统中有多少个服务器，分别安装了什么操作系统和软件；各设备之间采用什么类型的连接线缆，分配了什么 IP 地址和子网掩码等。而广域网和互联网对于具体的使用者是完全透明的，这也是我们经常看到一些系统架构图用一个云状的图形来表示广域网和互联网的原因。

虽然云状的图形中包含了许许多多的交换机、路由器、防火墙和服务器，但对广域网、互联网用户来讲，这些都是不需要知道的。这个云状图形代表的是广域网和互联网带给大家的互连互通的网络服务，无论我们在任何地方，通过一个网络接入线缆和用户名、密码，就可以接入广域网和互联网，享受网络带给我们的服务。

在存储的快速发展过程中，不同的厂商对云存储提供了不同的结构模型，在这里，我们介绍一个比较有代表性的云存储结构模型，这个模型的结构如图 4.1 所示。

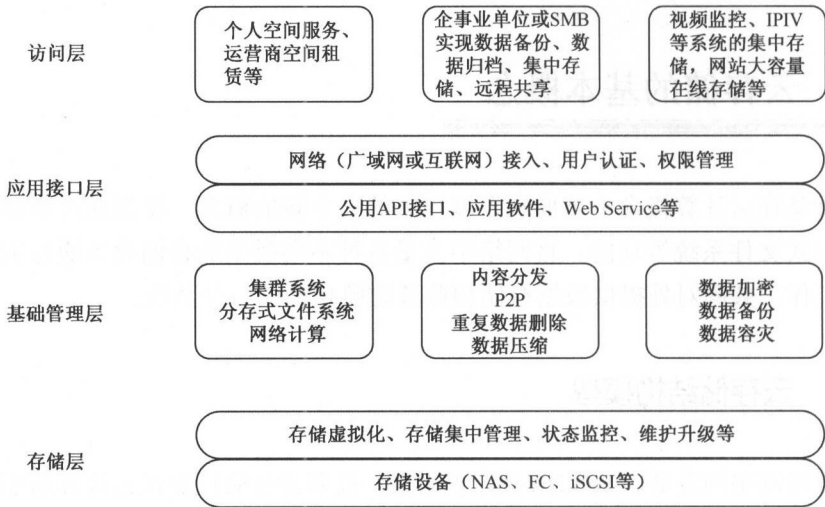


图 4.1 云存储系统的结构模型

云存储系统的结构模型自底向上由4层组成，分别为存储层、基础管理层、应用接口层、访问层。

1. 存储层

存储层是云存储最基础的部分。存储设备可以是FC（Fibre Channel）光纤通道存储设备，可以是NAS（Network Attached Storage，网络附属存储）和iSCSI（Internet Small Computer System Interface，Internet 小型计算机系统接口）等IP存储设备，也可以是SCSI（Small Computer System Interface，小型计算机系统接口）或SAS（Serial Attached SCSI，串行连接SCSI接口）等DAS（Direct Attached Storage，直接附加存储）存储设备。云存储中的存储设备往往数量庞大且分布在不同地域，彼此之间通过广域网、互联网或者FC光纤通道网络连接在一起。

存储设备之上是一个统一存储设备管理系统，可以实现存储设备的逻辑虚拟化管理、多链路冗余管理，以及硬件设备的状态监控和故障维护。

2. 基础管理层

基础管理层是云存储最核心的部分，也是云存储中最难以实现的部分。基础管理层通过集群、分布式文件系统和网格计算等技术，实现云存储中多个存储设备之间的协同工作，使多个存储设备可以对外提供同一种服务，并提供更大、更强、更好的数据访问性能。

CDN内容分发系统保证用户在不同地域访问数据的及时性，数据加密技术保证云存储中的数据不会被未经授权的用户所访问，同时，通过各种数据备份和容灾技术和措施可以保证云存储中的数据不会丢失，保证云存储自身的安全和稳定。

3. 应用接口层

应用接口层是云存储最灵活多变的部分。用户通过应用接口层实现对云端数据的存取操作，云存储更加强调服务的易用性。不同的云存储运营单位可以根据实际业务类型，开发不同的应用服务接口，提供不同的应用服务。服务提供商可以根据自己的实际业务需求，为用户开发相应的接口，比如视频监控应用平台、IPTV和视频点播应用平台、网络硬盘应用平台、远程数据备份应用平台等。

4. 访问层

经过身份验证或者授权的用户都可以通过标准的公用应用接口来登录云存储系统，享受云存储提供的服务。访问层的构建一般都遵循友好化、简便化和实用化的原则。访问层的用户通常包括个人数据存储用户、企业数据存储用户和服务集成商等。目前商用云存储系统对于中小型用户具有较大的性价比优势，尤其适合处于快速发展阶段的中小型企业。而由于云存储运营单位的不同，云存储提供的访问类型和访问手段也不尽相同。

尽管云存储有这样四层结构的划分，并且有一些尖端的技术也正处在研发阶段，例如EMC所宣布的道里（Daoli）可信基础架构项目，旨在提供可信的云计算平台，使用虚拟化

和可信计算技术，支持对单个主机计算机环境进行隔离，使之适合租借给多用户。简单说，道里项目可解决云计算下的安全问题，但是现有存储产品和技术已经足以支撑企业内部云存储服务需求。EMC 中国研发中心首席架构师任宇翔提出，云存储应该拥有几个基本的特征：一是大容量，云存储的最大存储容量可达数 PB。二是低成本，以 Google 为例，为了降低存储的采购和运维成本，它们的存储系统通常是自己“攒”的。三是灵活的扩展能力。他指出，云存储是存储技术的集大成者，虚拟化、数据压缩、重复数据删除、安全、基于策略的管理等都是云存储应该具备的能力。

4.1.2 云存储与传统存储系统的区别

用户使用云存储，并不是使用某一个存储设备，而是使用整个云存储系统带来的一种数据访问服务。如果用一句话来概括云存储与传统存储的区别的话，那就是：云存储不是存储，而是一种服务。

云存储系统需要存储的文件将随着用户数量的增长和存储内容的增加而呈指数级增长态势，这就要求存储系统的容量扩展能够跟得上数据量的增长，做到无限扩容，同时在扩展过程中最好还要做到简便易行，不能影响到数据中心的整体运行，也就是说，数据中心的存储系统容量的变化对于普通的数据服务使用者来说是透明的，即存储硬件的增减都不会影响到数据的访问，如果容量的扩展需要复杂的操作，甚至停机，这无疑会极大地降低数据中心的运营效率。

云时代的存储系统需要的不仅仅是容量的提升，对于性能的要求同样迫切，与以往只面向有限的用户不同，在云时代，存储系统将面向更为广阔的用户群体，用户数量级的增加使得存储系统也必须在吞吐性能上有飞速的提升，只有这样才能对请求做出快速的反应，这就要求存储系统能够随着容量的增加而拥有线性增长的吞吐性能，这显然是传统的存储架构无法达成的目标。

传统的存储系统由于没有采用分布式的文件系统，无法将所有访问压力平均分配到多个存储节点，因而在存储系统与计算系统之间存在着明显的传输瓶颈，由此而带来单点故障等多种后续问题，而集群存储正好解决了这一问题，给传统存储系统所面临的困难带来了一线生机。

要了解云存储系统与传统存储系统的区别，那就得搞清楚传统的存储系统在实际生产环境中所遇到的问题。显然，随着数据量的增多，传统的存储系统在下面这些问题的解决上越来越显得力不从心。

(1) 性能问题

由于数据量的激增，数据的索引效率也变得越来越为人们关注。而动辄上 TB 的数据。甚至是几百 TB 的数据，在索引时往往需要花上几分钟的时间。

传统的存储技术是把所有数据都当做对企业同等重要和同等有用的数据来进行处理，所有的数据集成到单一的存储体系之中，以满足业务持续性需求。但是在面临大数据时就

显得捉襟见肘了。

(2) 成本激增

在大型项目中，前端信息采集点过多，单台服务器承载量有限，就造成需要配置几十台，甚至上百台服务器的状况，这就必然导致建设成本、管理成本、维护成本、能耗成本的急剧增加。

(3) 磁盘碎片问题

视频监控系统往往采用回滚写入方式，这种无序的频繁读写操作，导致了磁盘碎片的大量产生。随着使用时间的增加，将严重地影响整体存储系统的读写性能，甚至导致存储系统被锁定为只读，而无法写入新的视频数据。

云存储系统与传统存储相比，则具有以下突出的优势。

(1) 量身定制

这个主要是针对私有云。云服务提供商专门为单一的企业客户提供一个量身定制的云存储服务方案，或者可以是企业自己的 IT 机构来部署一套私有云服务架构。私有云不但能为企业用户提供最优质的贴身服务，而且还能在一定程度上降低安全风险。亚马逊 S3 和 OpenStack 都能提供私有云环境。

(2) 成本低

就目前来说，企业在数据存储上所付出的成本是相当大的，而且这个成本还在随着数据的暴增而不断增加。为了减少这一成本压力，许多企业将大部分数据转移到云存储上，让云存储服务提供商来为他们解决数据存储的问题。这样就能花很少的价钱获得最优的数据存储服务。提供这些服务的企业，有 AWS S3、Windows Azure、国内的阿里云等。

(3) 管理方便

其实这一项也可以归纳为成本上的优势。因为将大部分数据迁移到云存储上后，所有的升级维护任务都由云存储服务提供商来完成，节约了企业存储系统管理员上的成本压力。还有就是云存储服务强大的可扩展性，当企业用户发展壮大后，突然发现自己先前的存储空间不足，就必须要考虑增加存储服务器来满足现有的存储需求。而云存储服务则可以很方便地在原有基础上扩展服务空间，满足需求。

4.2 云存储关键技术

为实现存储的低成本、高可扩展与资源池化，云存储技术应运而生，这其中最关键的是存储虚拟化技术与分布式存储技术的应用。

4.2.1 存储虚拟化技术

随着存储的需求不断增长，对于企业来说，所需要的存储服务器和磁盘都会随之相应

地快速增长。面对这种存储管理困境，存储虚拟化就是其中一种可选的解决方案。

那么，存储虚拟化的定义是什么呢？权威机构——SNIA（Storage Network Industry Association，全球网络存储工业协会）给出了以下定义：“通过将存储系统/子系统的内部功能从应用程序、计算服务器、网络资源中进行抽象、隐藏或隔离，实现独立于应用程序、网络的存储与数据管理”。

存储虚拟化技术的实现手段是通过将底层存储设备进行抽象化统一管理，底层硬件的异构性、特殊性等特性都被屏蔽了，对于服务器层来说只保留其统一的逻辑特性，从而实现了存储系统资源的集中，提供方便、统一的管理。存储虚拟化可以使管理员将不同的存储作为单个集合的资源来进行识别、配置和管理，存储资源的调度、存储设备的增减对于用户来说都是透明的。存储虚拟化是存储整合的一个重要组成部分，它能减少管理问题，而且能够提高存储利用率，这样可以降低新增存储的费用。

存储虚拟化与传统的存储相比，有什么不一样的地方吗？答案是肯定的。第一个区别：存储虚拟化相对于传统存储最大的优势在于磁盘的利用率很高。传统的存储磁盘利用率很低，大概只有 30%~70%，而采用了虚拟存储技术之后，磁盘的利用率能提高到 70%~90%，对于存储资源如此宝贵的企业来说，虚拟存储技术对他们的吸引力还是很高的。第二个区别是在存储的灵活性上，虚拟化的优点在于它可以把不同厂商生成的不同型号的异构的存储平台整合进来，适应异构环境，能够为资源的存储管理带来更好的灵活性。第三个区别是管理方便，它提供了一个大容量存储系统集中管理的手段，避免了由于存储设备扩充所带来的管理方面的麻烦。第四个区别是性能更好，虚拟化存储系统可以很好地进行负载均衡，把每一次数据访问所需的带宽合理地分配到各个存储模块上，提高了系统的整体访问带宽。

虚拟化存储根据在 I/O 路径中实现虚拟化的位置不同，可以分为 3 种实现技术：主机的虚拟存储、网络的虚拟存储以及存储设备的虚拟存储。

下面对 3 种存储虚拟化技术的实现以及它们的优缺点做简要介绍。

1. 基于主机的虚拟化存储技术

基于主机的虚拟化存储实现的核心技术是增加一个运行在操作系统下的逻辑卷管理软件，这个软件的功能是将磁盘上的物理块号映射成逻辑卷号，并以此把多个物理磁盘阵列映射成一个统一的虚拟的逻辑存储空间（逻辑块）实现存储虚拟化的控制和管理。从技术实施层面看，基于主机的虚拟化存储不需要额外的硬件支持，便于部署，只通过软件即可实现对不同存储资源的存储管理。但是，虚拟化控制软件也导致了此项技术的主要缺点：首先，软件的部署和应用影响了主机性能；其次，各种与存储相关的应用通过同一个主机，存在越权访问的数据安全隐患；最后，通过软件控制不同厂家的存储设备存在额外的资源开销，进而降低系统的可操作性与灵活性。

2. 基于网络的虚拟化技术

基于存储网络的虚拟化技术的核心是在存储区域网中增加虚拟化引擎实现存储资源的

集中管理,其具体实施一般通过具有虚拟化支持能力的路由器或交换机实现。在此基础上,存储网络虚拟化又可以分为带内虚拟化与带外虚拟化两类,二者主要的区别在于:带内虚拟化使用同一数据通道传送存储数据和控制信号,而带外虚拟化使用不同的通道传送数据和命令信息。基于存储网络的存储虚拟化技术架构合理,不占用主机和设备资源;但是其存储阵列中设备的兼容性需要严格验证,与基于设备的虚拟化技术一样,由于网络中存储设备的控制功能被虚拟化引擎所接管,导致存储设备自带的高级存储功能将不能使用。

3. 基于存储设备的虚拟存储技术

存储设备虚拟化技术依赖于提供相关功能的存储设备的阵列控制器模块,常见于高端存储设备,其主要应用针对异构的 SAN (Storage Area Network, 存储区域网络) 存储构架。此类技术的主要优点是不占主机资源,技术成熟度高,容易实施;缺点是核心存储设备必须具有此类功能,且消耗存储控制器的资源,同时由于异构厂家磁盘阵列设备的控制功能被主控设备的存储控制器接管导致其高级存储功能将不能使用。

4.2.2 分布式存储技术

除了虚拟存储技术以外,还有一种云存储技术称为分布式存储技术。由于分布式存储技术出现的时间相对于传统存储来说比较晚,因此,分布式存储相比传统的集中阵列存储设备,其技术和解决方案来说还处于发展的初级阶段,总体来看只具备部分场景下的存储需求实现能力。但是从发展趋势来看,通过一个可扩展的网络连接各离散的处理单元的分布式存储系统,其高可扩展性、低成本、无接入限制等优点是现有存储系统所无法比拟的。

分布式存储技术是指运用网络存储技术、分布式文件系统、网络存储技术等多种技术,实现云存储中的多种存储设备、多种应用、多种服务的协同工作。

网络存储技术将数据的存储从传统的服务器存储转移到网络设备存储。网络存储技术中比较典型的有直接附加存储 DAS、网络附加存储 NAS、存储区域网络 SAN。

分布式文件系统是指文件系统管理的物理存储资源并不一定直接连接在本地节点上,而是通过网络与网络节点互连。分布式文件系统可以将负载由单个节点转移到多个节点,常见的比较典型的分布式文件系统如 GFS 与 HDFS,存储在其中的每个文件都有 3 份拷贝,这 3 份拷贝位于不同的节点上,通过文件系统的控制,可以将数据的访问负载均衡到其他机器上,这样既能提高文件的读取效率,又能使整个文件系统处于一种均衡的状态,机器的利用率得以提升。分布式文件系统还可以避免由于单点失效而造成的整个系统崩溃。

网络存储具备更高的容错和冗余度,在负载出现波动的情况下可以保持高性能。网络存储技术具备先进的异构性、透明访问性、协同性、自主控制性和全生命周期性等特性。用户在使用网格的时候,可以不用关心存储容量、数据格式、数据安全性、数据读取位置和数据是否会丢失等问题。

面对云计算浪潮的来袭,大数据的存储向分布式文件系统提出了新的需求。由于互联

网应用的不断发展，本地文件系统由于单个节点本身的局限性，已经很难满足海量数据存取的需求了，因而不得不借助分布式文件系统，把系统负载转移到多个节点上。传统的分布式文件系统（如 NFS）中，所有数据和元数据存放在一起，通过单一的存储服务器提供。这种模式一般称为带内模式（In-band Mode）。随着客户端数目的增加，服务器就成了整个系统的瓶颈。因为系统所有的数据传输和元数据处理都要通过服务器，不仅单个服务器的处理能力有限，存储能力受到磁盘容量的限制，吞吐能力也受到磁盘 I/O 和网络 I/O 的限制。在当今对数据吞吐量要求越来越大的互联网应用中，传统的分布式文件系统已经很难满足应用的需要。

于是，一种新的分布式文件系统的结构出现了，那就是利用存储区域网络（SAN）技术，将应用服务器直接和存储设备相连接，大大提高数据的传输能力，减少数据传输的延时。在这样的结构里，所有的应用服务器都可以直接访问存储在 SAN 中的数据，而只有关于文件信息的元数据才经过元数据服务器处理提供，减少了数据传输的中间环节，提高了传输速率，减轻了元数据服务器的负载。每个元数据服务器可以向更多的应用服务器提供文件系统元数据服务。这种模式一般称为带外模式（Out-of-band Mode）。最近的 Storage Tank、CXFS、Lustre、BWFS 等都采用这样的结构，大名鼎鼎的 Hadoop 分布式文件系统也是这种结构，因此它们可以取得更好的性能和扩展性。区分带内模式和带外模式的主要依据是，关于文件系统元数据操作的控制信息是否和文件数据一起都通过服务器转发传送。前者需要服务器转发，后者是直接访问。随着 SAN 和 NAS 两种体系结构的成熟，越来越多的研究人员考虑如何结合这两种结构的优势，来创造更好的分布式文件系统。各种应用对存储系统提出了更多的需求。

- ① 大容量：现在的数据量比以前任何时期更多，生成的速度更快。
- ② 高性能：数据访问需要更高的带宽。
- ③ 高可用性：不仅要保证数据的高可用性，还要保证服务的高可用性。
- ④ 可扩展性：应用在不断变化，系统规模也在不断变化，这就要求系统提供很好的扩展性，并在容量、性能、管理等方面都能适应应用的变化。
- ⑤ 可管理性：随着数据量的飞速增长，存储的规模越来越庞大，存储系统本身也越来越复杂，这给系统的管理、运行带来了很高的维护成本。
- ⑥ 按需服务：能够按照应用需求的不同提供不同的服务，如不同的应用、不同的客户端环境、不同的性能等。

4.3 云存储系统分类

上一节我们讲了云存储系统的实现所采用的关键技术，依靠这些技术支持，市场上也已经出现了很多成熟的云存储服务。如何对这些存储系统做个分类呢？按照云存储资源的所有者来说，可以分为公共云存储、私有云存储和混合云存储 3 类。

1. 公共云存储

公共云存储是云存储提供商推出的付费使用的存储工具。云存储服务提供商建设并管理存储基础设施，集中空间来满足多用户需求，所有的组件放置在共享的基础存储设施里，设置在用户端的防火墙外部，用户直接通过安全的互联网连接访问。在公共云存储中，通过为存储池增加服务器，可以很快和很容易地实现存储空间增长。

公共云存储服务多是收费的，如亚马逊等公司都提供云存储服务，通常根据存储空间来收取使用费。用户只需要开通账号使用，不用了解任何云存储方面的软硬件知识或掌握相关技能。

2. 私有云存储

私有云存储顾名思义，它多是独享的云存储服务，为某一企业或社会团体独有。私有云存储建立在用户端的防火墙内部，并使用其所拥有或授权的硬件和软件。企业的所有数据保存在内部并且被内部 IT 员工完全掌握，这些员工可以集中存储空间来实现不同部门的访问或被企业内部的不同项目团队使用，无论其物理位置在哪儿。

私有云存储可由企业自行建立并管理，也可由专门的私有云服务公司根据企业的需要提供解决方案协助建立并管理。

私有云存储的使用成本较高，企业需要配置专门的服务器，获得云存储系统及相关应用的使用授权，同时还须支付系统的维护费用。

3. 混合云存储

混合云存储就是把公共云存储和私有云存储结合在一起。

混合云存储把公共云存储和私有云存储整合成更具功能性的解决方案。而混合云存储的“秘诀”就是处于中间的连接技术。为了更加高效地连接外部云和内部云的计算和存储环境，混合云解决方案需要提供企业级的安全性、跨云平台的可管理性、负载/数据的可移植性以及互操作性。

混合云存储主要用于按客户要求的访问，特别是需要临时配置容量的时候。从公共云上划出一部分容量配置一种私有或内部云可以帮助公司面对迅速增长的负载波动或高峰。尽管如此，混合云存储带来了跨公共云和私有云分配应用的复杂性。

另外，站在数据访问者的角度来看，分布式文件系统可以根据接口类型分成块存储、对象存储和文件存储这三类。如 Ceph 具备块存储、文件存储和对象存储的能力，GlusterFS 支持对象存储和文件存储的能力，而 MogileFS 只能作为对象存储并且通过 key 来访问。本节将针对每个技术分类进行详细介绍，并结合相应分类的代表性系统进行具体阐述。

4.3.1 分布式文件存储

分布式文件存储是云存储的一项关键技术，我们将从分布式文件系统存储的特点和其

中的关键技术入手，再结合一个典型的分布式文件系统 GFS 来全面介绍。

文件存储系统可提供通用的文件访问接口，如 POSIX、NFS、CIFS、FTP 等，实现文件与目录操作、文件访问、文件访问控制等功能。目前的分布式文件系统存储的实现有软硬件一体和软硬件分离两种方式，主要通过 NAS 虚拟化，或者基于 x86 硬件集群和分布式文件系统集成在一起，以实现海量非结构化数据处理能力。

软硬件一体方式的实现基于 x86 硬件，利用专有的、定制设计的硬件组件，与分布式文件系统集成在一起，以实现目标设计的性能和可靠性目标，产品代表有 Isilon、IBM SONAS GPFS。软硬件分离方式的实现基于开源分布式文件系统对外提供弹性存储资源，可采用标准 PC 服务器硬件，Hadoop 的 HDFS 就是典型开源分布式文件系统。

1. 分布式文件存储的概念

(1) 分布式文件系统的概念

说到分布式文件系统，不得不先提及文件系统。众所周知，文件系统是操作系统的一个重要组成部分，通过对操作系统所管理的存储空间的抽象，向用户提供统一的、对象化的访问接口，屏蔽对物理设备的直接操作和资源管理。如果没有文件系统，让用户直接与计算机存储硬件交互，这种方式的效率和可行性简直令人难以想象。

根据计算环境和所提供功能的不同，文件系统可划分为四个层次，从低到高依次是：单处理器单用户的本地文件系统，如 DOS 的文件系统；多处理器单用户的本地文件系统，如 OS/2 的文件系统；多处理器多用户的本地文件系统，如 UNIX 的本地文件系统；多处理器多用户的分布式文件系统，如 Lustre 文件系统。

本地文件系统（Local File System）是指文件系统管理的物理存储资源直接连接在本地节点上，处理器通过系统总线可以直接访问。分布式文件系统（Distributed File System）是指文件系统管理的物理存储资源不一定直接连接在本地节点上，而是通过计算机网络与节点相连。分布式文件系统的设计基于 C/S 模式。一个典型的分布式文件系统服务网络可能包括多个可以同时供多个用户访问的服务器。另外，网络节点的对等特性允许一些系统扮演客户机和服务器的双重角色。也就是说，一个节点既可以是一个服务器节点，同时也可以是一个客户机节点。这种概念在 P2P 网络中是常见的。举个例子来说，用户可以“发表”一个允许其他客户机访问的目录，这时候，如果有其他用户访问这个目录，那么这个目录对客户机来说就像一个服务器终端，可以像访问本地文件系统一样访问文件目录。

(2) 分布式文件系统存储的特点

在前面介绍分布式存储技术时提到了分布式存储系统的需求，那么分布式文件存储实现的时候就应该充分考虑这些需求，分布式文件存储具有以下特点。

① 扩展能力：毫无疑问，扩展能力是一个分布式文件存储最重要的特点。分布式文件系统存储中元数据管理一般是扩展的重要问题，GFS 采用元数据中心化管理，然后通过 Client 暂存数据分布来减小元数据的访问压力。GlusterFS 采用无中心化管理，在客户端采用一定的算法来对数据进行定位和获取。

② 高可用性：在分布式文件系统中，高可用性包括两层含义，一是整个文件系统的可用性，二是数据的完整和一致性。整个文件系统的可用性是分布式系统的设计问题，类似于 NoSQL 集群的设计，比如中心分布式系统的 Master 服务器、网络分区等。数据完整性则通过文件的镜像和文件自动修复等手段来解决，另外，部分文件系统如 GlusterFS 可以依赖底层的本地文件系统提供一定支持。

③ 协议和接口：分布式文件系统提供给应用的接口多种多样，如 HTTP RestFul 接口、NFS 接口、FTP 等 POSIX 标准协议，另外通常会有自己的专用接口。

④ 弹性存储：可以根据业务需要灵活地增加或缩减数据存储以及增删存储池中的资源，而不需要中断系统运行。弹性存储的最大挑战是减小或增加资源时的数据震荡问题。

⑤ 压缩、加密、去重、缓存和存储配额：这些功能的提供往往考验一个分布式文件系统是否具有可扩展性，一个分布式文件系统如果能方便地进行功能的添加而不影响总体的性能，那么这个文件系统就是良好的设计。这点 GlusterFS 就做得非常好，它利用类似 GNU/Hurd 的堆栈式设计，可以让额外的此类功能模块非常方便地增加。另外压缩在一定程度上减少了文件传输时的带宽消耗。加密为文件和文件夹提供安全保障。

2. 分布式文件存储实例

2003 年，Google 公开了他们的分布式文件系统的设计思想，引起了业内轰动。在本书的第 3 章中，具体介绍了 Google 的这项伟大发明，本节回顾一下 GFS 的主要设计思想。

Google File System 是一个可扩展的分布式文件系统，用于大型的、分布式的、对海量数据进行访问的应用。它运行于廉价的普通硬件上，但提供了容错复制功能，可以给大量的用户提供总体性能较高的可靠服务。

1) GFS 的设计观点

GFS 与过去的分布式文件系统有很多相同的目标，比如性能、可扩展性、可靠性、可用性，但 GFS 的设计受到了当前及预期的应用方面的工作量及技术环境的驱动，这反映了它与早期的文件系统明显不同的设想。需要对传统的选择进行重新检验并进行完全不同的设计观点的探索。

GFS 与以往的文件系统的不同的观点如下。

① 组件错误（包括存储设备或存储节点的故障）不再被当做异常，而是将其作为常见的情况加以处理。因为文件系统由成百上千个用于存储的普通计算机构成，而这些机器由廉价的普通部件组成，但却面向众多的数据访问者。俗话说得好，一分钱一分货，廉价部件用得多了，质量就堪忧了，因此一些机器随时都有可能无法工作甚至还有无法恢复的可能。所以实时监控、错误检测、容错、自动恢复对系统来说必不可少。

② 按照传统的标准，文件都非常大。长度达几个 GB 的文件是很平常的。每个文件通常包含很多应用对象。当经常要处理快速增长的、包含数以万计对象的数据集时，即使底层文件系统提供支持，我们也很难管理成千上万的 KB 规模的文件块。因此在设计中，操作的参数、块的大小必须要重新考虑。对大型的文件的管理一定要能做到高效，对小型的

文件也必须支持，但不必优化。

③ 大部分文件的更新是通过添加新数据完成的，而不是改变已存在的数据。在一个文件中随机的操作在实践中几乎不存在。一旦写完，文件就只可读，很多数据都有这些特性。一些数据可能组成一个大仓库以供数据分析程序扫描。有些是运行中的程序连续产生的数据流。有些是档案性质的数据，有些是在某个机器上产生、在另外一个机器上处理的中间数据。由于这些对大型文件的访问方式，添加操作成为了性能优化和原子性保证的焦点。而在客户机中缓存数据块则失去了吸引力。

④ 工作量主要由两种读操作构成：对大量数据的流方式的读操作和对少量数据的随机方式的读操作。在前一种读操作中，可能要读几百 KB，通常达 1MB 和更多。根据局部性原理，来自同一个客户的连续操作通常会读文件的一个连续的区域。随机的读操作通常在一个随机的偏移处读几个 KB。性能敏感的应用程序通常将对少量数据的读操作进行分类并进行批处理以使得读操作稳定地向前推进，而不要让它来来回回地读。

⑤ 工作量还包含许多对大量数据进行的连续的向文件添加数据的写操作。所写的数据的规模和读相似。一旦写完，文件很少改动。在随机位置对少量数据的写操作也支持，但不必非常高效。

2) GFS 的设计策略

在了解了 GFS 与以往文件系统的不同观点之后，我们接下来重点分析它的设计策略。由于 GFS 最初是用来存储大量的网页的，而且这些数据一般都是一次写入多次读取的，那在设计文件系统的时候就要特别考虑该如何进行设计了，主要体现在以下几个方面。

① 一个 GFS 集群由一个 Master 和大量的 ChunkServer 构成，并被许多客户（Client）访问。文件被分成固定大小的块。每个块由一个不变的、全局唯一的 64 位的 chunk-handle 标识，chunk-handle 是在块创建时由 Master 分配的。

② 出于可靠性考虑，每一个块被复制到多个 ChunkServer 上。默认情况下，保存 3 个副本，但这可以由用户指定。这些副本在 Linux 文件系统上作为本地文件存储。

③ 每个 GFS 集群只有一个 Master，维护文件系统所有的元数据（metadata），包括名字空间、访问控制信息、从文件到块的映射以及块的当前位置。它也控制系统范围的活动，如块租约（lease）管理、孤儿块的垃圾收集、ChunkServer 间的块迁移。

④ Master 定期通过 HeartBeat 消息与每一个 ChunkServer 通信，给 ChunkServer 传递指令并收集它的状态。

⑤ 客户和 ChunkServer 都不缓存文件数据。因为用户缓存数据几乎没有什么作用，这是由于数据太多或工作集太大而无法缓存。不缓存数据简化了客户程序和整个系统，因为不必考虑缓存的一致性问题。但用户缓存元数据（metadata）。此外，ChunkServer 也不必缓存文件，因为块是作为本地文件存储的。依靠 Linux 本身的缓存 Cache 在内存中保存数据。

3) GFS 的架构和组件

GFS 的整体架构在第 3 章已经介绍过，其组件主要有两个：Master 和 ChunkServer。

(1) Master

Master 的功能和作用如下。

① 保存文件/Chunk 名字空间、访问控制信息、文件到块的映射以及块的当前位置，全内存操作（64 字节每 Chunk）。

② Chunk 租约管理、垃圾和孤儿 Crunk 回收、不同服务器间的 Chunk 迁移。

③ 记录操作日志：操作日志包含了对 metadata 所做的修改的历史记录。它作为逻辑时间定义了并发操作的执行顺序。文件、块以及它们的版本号都由它们被创建时的逻辑时间而唯一、永久地被标识。

④ 在多个远程机器备份 Master 数据。

⑤ 设置 Checkpoint，用于快速恢复。

(2) ChunkServer 的一些特性

① Chunk（数据块）的大小被固定为 64MB，这个尺寸相对来说还是挺大的，这是因为较大的 Chunk 尺寸能够减少元数据访问的开销，减少同 Master 的交互。

② Chunk 位置信息并不是一成不变的，可能会由于系统的负载均衡、机器节点的增减而动态改变。

块规模是设计中的一个关键参数，GFS 选择的是 64MB，这比一般的文件系统的块规模要大得多。每个块的副本作为一个普通的 Linux 文件存储，在需要的时候可以扩展。块规模较大的好处如下。

① 减少 Client 和 Master 之间的交互。在开始读取文件之前，客户端需要向 Master 请求块位置信息，对于读写大型文件这种减少尤为重要。即使对于访问少量数据的随机读操作也可以很方便地为一个规模达几个 TB 的工作集缓存块位置信息。

② Client 在一个给定的块上很可能执行多个操作，和一个 ChunkServer 保持较长时间的 TCP 连接可以减少网络负载。

③ 这减少了 Master 上保存的元数据（metadata）的规模，从而使得可以将 metadata 放在内存中。这又会带来一些别的好处。

但是块规模较大也有不利的一面：

① Chunk 较大可能产生内部碎片。

② 同一个 Chunk 中存在许多小文件可能产生访问热点，一个小文件可能只包含一个块，如果很多 Client 访问该文件的话，存储这些块的 ChunkServer 将成为访问的热点。但在实际应用中，应用程序通常顺序地读包含多个块的文件，所以这不是一个主要问题。

3. GFS 的容错和诊断

GFS 为文件系统提供了很高的容错能力，主要体现在两个方面：高可靠性和数据完整性。

(1) 高可靠性

① 快速恢复。不管如何终止服务，Master 和数据块服务器都会在几秒内恢复状态和运

行。实际上，并不对正常终止和不正常终止进行区分，服务器进程都会被切断而终止。客户机和其他的服务器会经历一个小小的中断，然后它们的特定请求超时，重新连接重启的服务器，重新请求。

② 数据块备份。每个数据块（Chunk）都会被备份到不同机架的不同服务器上，通常是每个数据块都有 3 个副本。对不同的名字空间，用户可以设置不同的备份级别。在数据块服务器掉线或数据被破坏时，Master 会按照需要来复制数据块。

③ Master 备份。为确保可靠性，Master 的状态、操作记录和检查点（check point）都在多台机器上进行了备份。一个操作只有在数据块服务器硬盘上刷新并被记录在 Master 和其备份上之后才算是成功的。如果 Master 或硬盘失败，系统监视器会发现并通过改变域名启动它的一个备份机，而客户机仅仅使用规范的名称来访问，并不会发现 Master 的改变。

（2）数据完整性

每个数据块服务器都利用校验和来检验存储数据的完整性。原因：每个服务器随时都有发生崩溃的可能性，并且在两个服务器间比较数据块也是不现实的，同时，在两台服务器间复制数据并不能保证数据的一致性。

每个 Chunk 按 64kB 的大小分成块，每个块有 32 位的校验和，校验和和日志存储在一起，和用户数据分开。在读数据时，服务器首先检查与被读内容相关部分的校验和，因此，服务器不会传播错误的数据。如果所检查的内容和校验和不符，服务器就会给数据请求者返回一个错误的信息，并把这个情况报告给 Master。客户机就会读其他的服务器来获取数据，而 Master 则会从其他的副本来复制数据，等到一个新的副本完成时，Master 就会通知报告错误的服务器删除出错的数据块。

附加写数据时的校验和计算优化了，因为这是主要的写操作。因此只是更新增加部分的校验和，即使末尾部分的校验和数据已被损坏而没有检查出来，新的校验和与数据会不相符，这种冲突在下次使用时将会被检查出来。

相反，如果是覆盖现有数据的写，在写以前，我们必须检查第一和最后一个数据块，然后才能执行写操作，最后计算和记录校验和。如果在覆盖以前不先检查首位数据块，计算出的校验和则会因为没被覆盖的数据而产生错误。

在空闲时间，服务器会检查不活跃的数据块的校验和，这样可以检查出不经常读的数据的错误。一旦错误被检查出来，服务器会复制一个正确的数据块来代替错误的。

4. GFS 的扩展性能

对于分布式文件系统存储来说，系统的可扩展性是系统设计好坏的一个非常关键的指标。由于 GFS 采用单一的 Master 的设计结构，因此扩展主要在于 ChunkServer 节点的加入，每当有 ChunkServer 加入的时候 Master 会询问其所拥有的块的情况，而 Master 在每次启动的时候也会主动询问所有 ChunkServer 的情况。

GFS 单一 Master 的设计方式使得系统管理简单、方便，但也有不利的一面：随着系统规模的扩大，单一 Master 是否会成为瓶颈？这看起来是限制系统可扩展性和可靠性的一个

缺陷，因为系统的最大存储容量和正常工作时间受制于主服务器的容量和正常工作时间，也因为它要将所有的元数据进行编制，并且因为几乎所有的动作和请求都经过它。

但是 Google 的工程师们辩解说事实并不是这样。元数据是非常紧凑的，仅仅只有数 KB 到数 MB 的大小，并且主服务器通常是网络上性能最好的节点之一；至于可靠性，通常有一个“影子”主服务器作为主服务器的镜像，一旦主服务器失败它将接替工作。另外，主服务器极少成为瓶颈，因为客户端仅仅取得元数据，然后会将它们缓存起来；随后的交互工作直接与 ChunkServer 进行。同样，使用单个主服务器可以大幅度地降低软件的复杂性，如果有多个主服务器，软件将变得复杂才能够保证数据完整性、自动操作、负载均衡和安全性。

5. 分布式文件存储小结

根据分布式文件系统存储的特点，并结合上面 GFS 的实例，我们总结一下分布式文件系统存储在设计、实现时主要关注的几个方面。

① 设计特点：分布式能力、性能、容灾、维护和扩展、成本。

② 分布式文件系统主要关键技术：全局名字空间、缓存一致性、安全性、可用性、可扩展性。

③ 其他关键技术：文件系统的快照和备份技术、热点文件处理技术、元数据集群的负载均衡技术、分布式文件系统的日志技术。

4.3.2 分布式块存储

1. 分布式块存储的概念

在讨论分布式块存储之前，我们先解释一下块存储的概念，块存储简单来说就是提供了块设备存储的接口，用户需要把块存储卷附加到虚拟机或者裸机上以与其交互。这些卷都是持久的，因为它们可以从运行实例上被解除或者重新附加而数据保持不变。

这样解释有些读者可能还不太了解什么是块存储，下面先从单机块设备工具开始介绍，让读者对块存储建立起初步的印象。简单来说，一个硬盘是一个块设备，内核检测到硬盘后，在 `/dev/` 下会看到 `/dev/sda/`。为了用一个硬盘得到不同的分区来做不同的事，我们使用 `fdisk` 工具得到 `/dev/sda1`、`/dev/sda2` 等。这种方式通过直接写入分区表来规定和切分硬盘，是比较原始的分区方式。庆幸的是有一些单机块设备工具能帮我们完成分区，其中 LVM 是一种逻辑卷管理器。通过 LVM 来对硬盘创建逻辑卷组和得到逻辑卷，要比 `fdisk` 方式更加弹性。LVM 基于 Device-mapper 用户程序实现。Device-mapper 是一种支持逻辑卷管理的通用设备映射机制，为存储资源管理的块设备驱动提供了一个高度模块化的内核架构。

在面对极具弹性的存储需求和性能要求下，单机或者独立的 SAN (Storage Area Networking) 越来越不能满足企业的需要。如同数据库系统一样，块存储在 `scale up` 的瓶颈下也面临着 `scale out` 的需要。我们可以用以下几个特点来描述分布式块存储系统的概念。

- ① 分布式块存储可以为任何物理机或者虚拟机提供持久化的块存储设备。
- ② 分布式块存储系统管理块设备的创建、删除和 attach/detach。
- ③ 分布式块存储支持强大的快照功能，快照可以用来恢复或者创建新的块设备。
- ④ 分布式存储系统能够提供不同 I/O 性能要求的块设备。

2. 分布式块存储实例

分布式块存储目前已经相对成熟，市场上也有很多基于分布式块存储技术实现的产品。在本节中，我们将结合几个市场上流行的产品进行介绍。

(1) Amazon EBS

Amazon 作为领先的 IaaS 服务商，其 API 目前是 IaaS 的事实标准。Amazon EC2 目前在大多数方面远超其他 IaaS 服务商。Amazon EBS 是专门为 Amazon EC2 虚拟机设计的弹性块存储服务。Amazon EBS 可以为 Amazon EC2 的虚拟机创建卷 volumes，Amazon EBS 卷类似没有格式化的外部卷设备。卷有设备名称，同时也提供了块设备接口。用户可以在 Amazon EBS 卷上驻留自己的文件系统，或者直接作为卷设备使用。EBS 定价为每月每 GB 容量 10 美分，或者每向卷发出 100 万次请求 10 美分。据 Amazon 称，用户还可以将虚拟机的数据以快照的方式存储到 Amazon 的 S3。

一般来说，可以创建多达 20 个 Amazon EBS 卷，卷的大小可从 1GB 到 1TB。在相同 Availability Zone 中，每个 Amazon EBS 卷可以被任何 Amazon EC2 虚拟机使用。如果需要超过 20 个卷，则需要提出申请。

同时，Amazon EBS 提供了快照功能。可以将快照保存到 Amazon S3 中，其中第一个快照是全量快照，随后的快照都是增量快照。可以使用快照作为新的 Amazon EBS 卷的起始点，这样当虚拟机数据受到破坏时可以选择回滚到某个快照来恢复数据，从而提高数据的安全性及可用性。

Amazon EC2 实例可以将根设备数据存储在 Amazon EBS 或者本地实例存储上。使用 Amazon EBS 时，根设备中的数据将独立于实例的生命周期保留下来，使得在停止实例后仍可以重新启动使用，与笔记本电脑关机并在再次需要时重新启动相似。另外，本地实例存储仅在实例的生命周期内保留。这是启动实例的一种经济方式，因为数据没有存储到根设备中。

EBS 可以在卷连接和使用期间实时拍摄快照。不过，快照只能捕获已写入 Amazon EBS 卷的数据，不包含应用程序或操作系统已在本地缓存的数据。如果需要确保能为实例连接的卷获得一致的快照，需要先彻底地断开卷连接，再发出快照命令，然后重新连接卷。

EBS 快照目前可以跨 regions 增量备份，意味着 EBS 快照时间会大大缩短，从另一方面增加了 EBS 使用的安全性。

下面通过 Amazon EBS 容错处理和使用快照加载新卷的过程来了解 Amazon EBS 的功能。

Amazon EBS 可以将任何的实例（即运行中的虚拟机）关联到卷。当一个实例失效，

Amazon EBS 卷可以自动地解除与失效节点的关联，从而可以将该卷关联到新的实例，如图 4.2 所示。步骤如下。

① 运行中的 Amazon EC2 实例被关联到 Amazon EBS 卷，而这个实例突然失效或者出现异常。

② 为了恢复该实例，解除 Amazon EBS 卷和实例的关系（如果没有自动解除），加载一个新的 Amazon EC2 实例，将其关联到 Amazon EBS 卷。

③ 在 Amazon EBS 卷失效的情况下（几率极低），可以根据快照创建一个新的 Amazon EBS 卷。

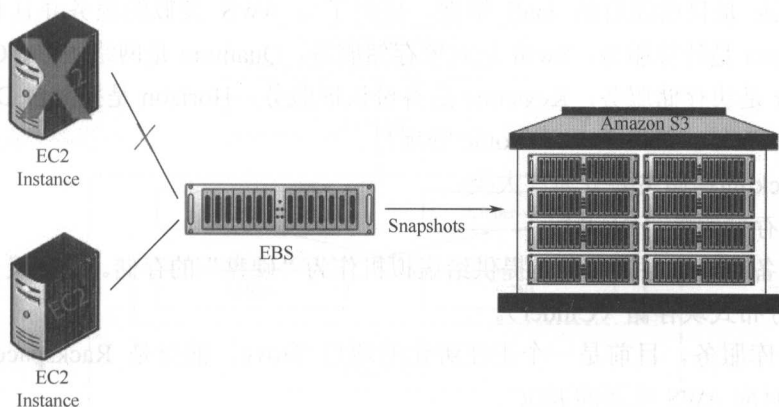


图 4.2 EBS 容错处理

我们可以使用 Amazon EBS 快照作为一个起点来加载若干个新卷（图 4.3）。加载过程如下。

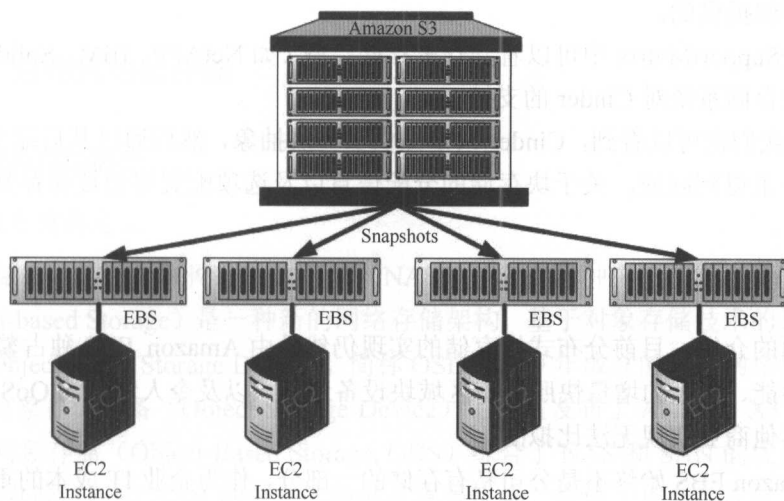


图 4.3 EBS 快照

① 假设现在有个大数据量的 Web Service 服务正在运行。

② 当数据都正常的时候,可以为自己的卷创建快照,并将这些快照存储在 Amazon S3 上。

③ 当服务数据剧增时,需要根据快照加载新的卷,然后启动新的实例,再将新的实例关联到新的卷。

④ 当服务下降时,可以关闭一个或多个 Amazon EC2 实例,并删除相关的 EBS 卷。

总的来说,Amazon EBS 是目前 IaaS 服务商最引人注目的服务之一,目前的 OpenStack、CloudStack 等其他开源框架都无法提供 Amazon EBS 的弹性和强大的服务。了解和使用 Amazon EBS 是学习 IaaS 块存储的最好手段。

(2) Cinder

OpenStack 是目前流行的 IaaS 框架,提供了与 AWS 类似的服务并且兼容其 API。OpenStack Nova 是计算服务,Swift 是对象存储服务,Quantum 是网络服务,Glance 是镜像服务,Cinder 是块存储服务,Keystone 是身份认证服务,Horizon 是控制台 Dashboard,另外还有 Heat、Oslo、Ceilometer、Ironic 等项目。

OpenStack 的存储主要分为三大类。

① 对象存储服务 (Swift)。

② 块设备存储服务,主要是提供给虚拟机作为“硬盘”的存储。这里又分为两块:本地块存储和分布式块存储 (Cinder)。

③ 数据库服务,目前是一个正在孵化的项目 Trove,前身是 Rackspace 开源出来的 RedDwarf,对应 AWS 里面的 RDC。

Cinder 是 OpenStack 中提供类似于 EBS 块存储服务的 API 框架。它并没有实现对块设备的管理和实际服务,而是为后端不同的存储结构提供了统一的接口,不同的块设备服务厂商在 Cinder 中实现其驱动支持以与 OpenStack 进行整合。后端的存储可以是 DAS、NAS、SAN、对象存储或者分布式文件系统。也就是说,Cinder 的块存储数据完整性、可用性保障是由后端存储提供的。

在 CinderSupportMatrix 中可以看到众多存储厂商(如 NetAPP、IBM、SolidFire、EMC)和众多开源块存储系统对 Cinder 的支持。

从图 4.4 我们也可以看到,Cinder 只是提供了一层抽象,然后通过其后端支持的 driver 实现发出命令来得到回应。关于块存储的分配信息以及选项配置等会被保存到 OpenStack 统一的 DB 中。

3. 分布式块存储小结

通过上面的介绍,目前分布式块存储的实现仍然是由 Amazon EBS 独占鳌头,其卓越稳定的读写性能、强大的增量快照和跨区域块设备迁移,以及令人惊叹的 QoS 控制都是目前开源或者其他商业实现无法比拟的。

不过 Amazon EBS 始终不是公司私有存储的一部分,作为企业 IT 成本的重要部分,块存储正在发生改变。EMC 发布了其 ViPR 平台,并开放了其接口,试图接纳其他厂商和开源实现。Nexenta 在颠覆传统的存储专有硬件,在其上软件实现原来只有 SDN 的能力,让企业客户完全摆脱存储与厂商的绑定。Inktank 极力融合 OpenStack 并推动 Ceph 在 OpenStack

社区的影响力。这些都说明了无论是目前的存储厂商还是开源社区都在极力推动整个分布式块存储的发展，存储专有设备的局限性正在进一步弱化原有企业的存储架构。

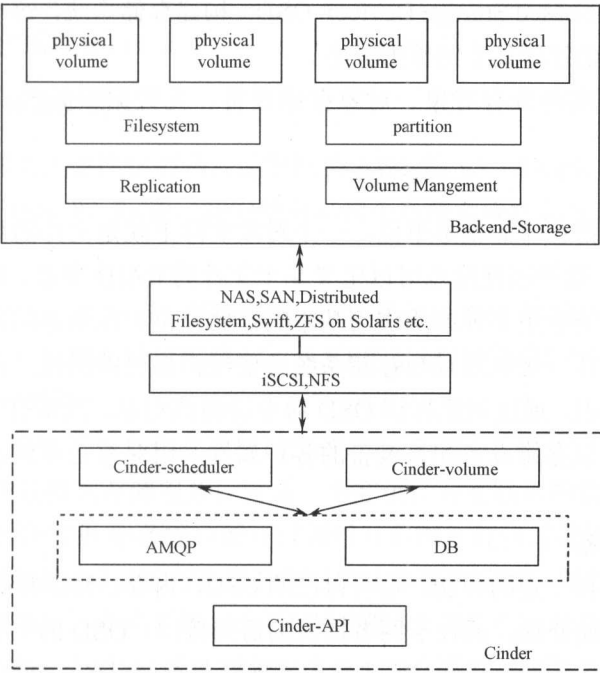


图 4.4 Cinder 架构图

在分布式块存储和 OpenStack 之间，可以打造更巩固的纽带，将块存储在企业私有云平台上做更好的集成和运维。

4.3.3 分布式对象存储

1. 对象存储的概念

1) 对象存储的定义

存储局域网（SAN）和网络附加存储（NAS）是目前两种主流网络存储架构，而对象存储（Object-based Storage）是一种新的网络存储架构，基于对象存储技术的设备就是对象存储设备（Object-based Storage Device），简称 OSD。1999 年成立的全球网络存储工业协会（SNIA）的对象存储设备（Object Storage Device）工作组发布了 ANSI 的 X3T10 标准。总体上来讲，对象存储（Object-Based Storage, OBS）综合了 NAS 和 SAN 的优点，同时具有 SAN 的高速直接访问和 NAS 的分布式数据共享等优势，提供了具有高性能、高可靠性、跨平台以及安全的数据共享的存储体系结构。

2) 对象存储的架构

对象存储的核心是将数据通路（数据读或写）和控制通路（元数据）分离，并且基于对象存储设备（Object-based Storage Device, OSD）构建存储系统，每个对象存储设备具有一定的智能，能够自动管理其上的数据分布。

对象存储结构组成部分有对象、对象存储设备、元数据服务器、对象存储系统的客户端。

(1) 对象

对象是系统中数据存储的基本单位，一个对象实际上就是文件的数据和一组属性信息（Meta Data）的组合，这些属性信息可以定义基于文件的 RAID 参数、数据分布和服务质量等，而传统的存储系统中用文件或块作为基本的存储单位，在块存储系统中还需要始终追踪系统中每个块的属性，对象通过与存储系统通信维护自己的属性。在存储设备中，所有对象都有一个对象标识，通过对象标识 OSD 命令访问该对象。通常有多种类型的对象，存储设备上的根对象标识存储设备和该设备的各种属性，组对象是存储设备上共享资源管理策略的对象集合等。

(2) 对象存储设备

对象存储设备具有一定的智能，它有自己的 CPU、内存、网络和磁盘系统，OSD 同块设备的不同不在于存储介质，而在于两者提供的访问接口。OSD 的主要功能包括数据存储和安全访问。目前国际上通常采用刀片式结构实现对象存储设备。OSD 提供 3 个主要功能。

① 数据存储。OSD 管理对象数据，并将它们放置在标准的磁盘系统上，OSD 不提供块接口访问方式，Client 请求数据时用对象 ID、偏移进行数据读写。

② 智能分布。OSD 用其自身的 CPU 和内存优化数据分布，并支持数据的预取。由于 OSD 可以智能地支持对象的预取，从而可以优化磁盘的性能。

③ 每个对象元数据的管理。OSD 管理存储在其上对象的元数据，该元数据与传统的 inode 元数据相似，通常包括对象的数据块和对象的长度。而在传统的 NAS 系统中，这些元数据是由文件服务器维护的，对象存储架构将系统中主要的元数据管理工作交由 OSD 来完成，降低了 Client 的开销。

(3) 元数据服务器（Metadata Server, MDS）

MDS 控制 Client 与 OSD 对象的交互，主要提供以下几个功能。

① 对象存储访问：MDS 构造、管理描述每个文件分布的视图，允许 Client 直接访问对象。MDS 为 Client 提供访问该文件所含对象的能力，OSD 在接收到每个请求时将先验证该能力，然后才可以访问。

② 文件和目录访问管理：MDS 在存储系统上构建一个文件结构，包括限额控制、目录和文件的创建和删除、访问控制等。

③ Client Cache 一致性：为了提高 Client 性能，在对象存储系统设计时通常支持 Client 方的 Cache。由于引入 Client 方的 Cache，带来了 Cache 一致性问题，MDS 支持基于 Client 的文件 Cache，当 Cache 的文件发生改变时，将通知 Client 刷新 Cache，从而防止 Cache 不

一致引发的问题。

(4) 对象存储系统的客户端

为了有效支持 Client 访问 OSD 上的对象，需要在计算节点实现对象存储系统的 Client，通常提供 POSIX 文件系统接口。

2. 分布式对象存储实例

分布式对象存储的代表性实例是云计算巨头 AWS 的 S3 (Simple Storage Service)，在开源界对应着 OpenStack 的 Swift，我们将对这两个系统做详细的分析。

1) AWS S3

Amazon Simple Storage Service 是亚马逊 AWS 服务在 2006 年第一个正式对外推出的云计算服务。下面结合我们的实际使用体验，从 S3 的数据结构和特点进行介绍。

(1) S3 的背景与概览

S3 为开发人员提供了一个高度扩展 (Scalability)、高持久性 (Durability) 和高可用 (Availability) 的分布式数据存储服务。它是一个完全针对互联网的数据存储服务，应用程序可以通过一个简单的 Web 服务接口就可以通过互联网在任何时候访问 S3 上的数据。当然用户存放在 S3 上的数据可以进行访问控制以保障数据安全性。这里所说的访问 S3 包括读、写、删除等多种操作。在刚开始接触 S3 时要把 S3 与我们日常所说的网盘区分开来，虽然都属于云存储范畴，但是 S3 是针对开发人员、主要通过 API 编程使用的一个服务，而网盘这样的云存储服务则提供了一个给最终用户使用的服务界面。虽然 S3 也可以通过 AWS 的 Web 管理控制台或命令行使用，但是 S3 主要针对开发人员，在理解上可以看成云存储的后台服务。比如，Dropbox 是很多人都喜欢使用的云存储服务，它就是一个典型的 AWS 客户，其所有的用户文件保存在 S3 中。图 4.5 显示了 S3 在过去几年中所存储的数据对象的增长情况。

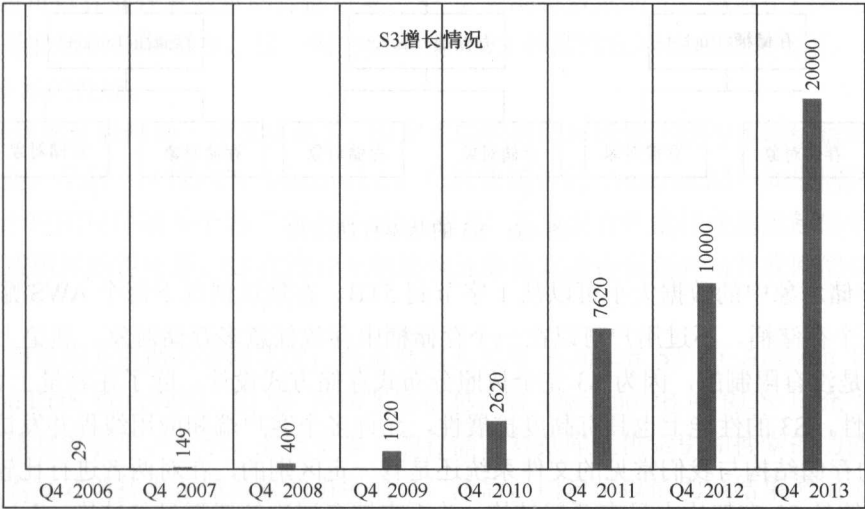


图 4.5 S3 近几年增长情况 (单位: 亿)

S3 云存储解决了大规模数据持久化存储的问题。前面提到 EBS 虽然是持久化的，但有容量限制，最大容量为 1TB。在这个信息爆炸的时代，如何保存海量数据成为一大难题。有了 S3 云存储后，用户可以把注意力集中到其他地方，更专注于业务而不用关心运维和容量规划。

(2) S3 的数据结构

S3 的数据存储结构非常简单，就是一个扁平化的两层结构：一层是存储桶（Bucket，又称存储段），另一层是存储对象（Object，又称数据元）。

存储桶是 S3 中用来归类数据的一个方式，它是存储数据的容器。每一个存储对象都需要存储在某一个存储桶中。存储桶是 S3 命名空间的最高层，它会成为用户访问数据的域名的一部分，因此存储桶的名字必须是唯一的，而且需要保持 DNS 兼容，比如采用小写、不能用特殊字符等。例如，我们创建了一个名为 cloud-uestc 的存储桶，那么对应的域名就是 cloud-uestc.s3.amazonaws.com，以后我们可以通过 http:// cloud-uestc.s3.amazonaws.com/来访问其中存储的数据。由于数据存储的地理位置有时对用户来说很重要，因此在创建存储桶的时候 S3 会提示选择区域（Region）信息。

存储对象就是用户实际要存储的内容，其构成就是对象数据内容再加上一些元数据信息。这里的对象数据通常是一个文件，而元数据就是描述对象数据的信息，比如数据修改的时间等。如果我们在 cloud-uestc 的存储桶中存放了一个文件 picture.jpg，那么我们可以通过 http:// cloud-uestc.s3.amazonaws.com/picture.jpg 这个 URL 来访问这个文件。从这个 URL 访问我们可以看到，存储桶名称需要全球唯一，而存储对象的命名则需要在存储桶中唯一。只有这样我们才能通过一个全球唯一的 URL 访问到指定的数据。S3 的数据存储结构如图 4.6 所示。

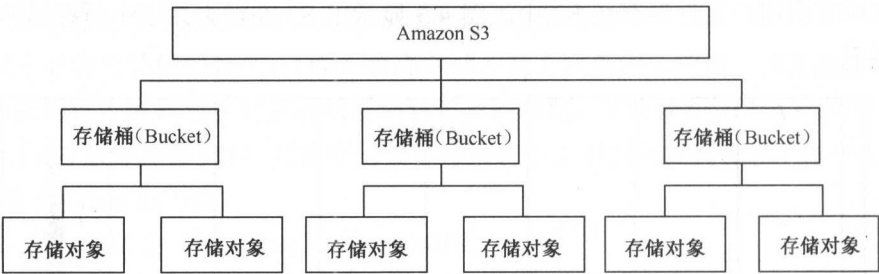


图 4.6 S3 的基本存储结构

S3 存储对象中的数据大小可以从 1 字节到 5TB。在默认情况下每个 AWS 账号最多能创建 100 个存储桶。不过用户可以在一个存储桶中存放任意多存储对象。理论上存储桶中的对象数是没有限制的，因为 S3 完全按照分布式存储方式设计。除了在容量上 S3 具有很高的扩展性，S3 的性能上也具有高度扩展性，允许多个客户端和应用线程并发访问数据。

S3 的存储结构与我们常见的文件系统还是有一定区别的，在对两者进行比较的时候，需要注意的是 S3 在架构上只有两层结构，并不支持多层次的树形目录结构。不过我们可以通过设计带 “/” 的存储对象名称来模拟出一个树形结构。例如有些 S3 工具就提供了一个

操作选项是“创建文件夹”，实际上就是通过控制存储对象的名称来实现的。

(3) S3 的特点

作为云存储的典型代表，Amazon S3 在扩展性、持久性和性能等几个方面有自己明显的特点。S3 云存储最大的特点是无限容量、高持久化、高可用，但它是一个 key-value 结构的存储。与 EBS 相比，它缺少目录结构，所以在用户的业务里，一般都会使用数据库保存 S3 云存储上数据的元信息。

① 耐久性和可用性。

为了保证数据的耐久性和可用性，用户保存在 S3 上的数据会自动地在选定地理区域中多个设施（数据中心）和多个设备中进行同步存储。S3 存储提供了 AWS 平台中最高级别的数据持久性和可用性。除了分布式的数据存储方式之外，S3 还内置了数据一致性检查机制来提供错误更正功能。S3 的设计不存在单点故障，可以承受两个设施同时出现数据丢失，因此非常适合用于任务关键型数据的主要数据存储。实际上，Amazon S3 旨在为每个存储对象提供 99.999999999%（11 个 9）的年持久性和 99.99% 的年可用性。除了内置冗余外，S3 还可通过使用 S3 版本控制功能使数据免遭应用程序故障和意外删除造成的损坏。对于可以根据需要轻松复制的非关键数据（如转码生成的媒体文件、镜像缩略图等），可以使用 Amazon S3 中的降低冗余存储（ReducedRedundancy Storage, RRS）选项。RRS 的持久性为 99.99%，当然它存储费用也更低。尽管 RRS 的持久性稍逊于标准 S3，但仍高出一般磁盘驱动器约 400 倍。

② 弹性和可扩展性。

Amazon S3 的设计能够自动提供高水平的弹性和扩展性。一般的文件系统可能会在一个目录中存储大量文件时遇到问题，但是 S3 能够支持在任何存储桶中无限量地存储文件。另外，与磁盘不同的是，磁盘大小会限制可存储的数据总量，而 Amazon S3 存储桶可以存储无限量的数据。在数据大小方面目前 S3 的唯一限制是单个存储对象的大小不能超过 5TB，但是可以存储任意数量的存储对象，S3 会自动将数据的冗余副本扩展和分发到同一地区内其他位置的服务器中，这一切完全通过 AWS 的高性能基础设施来实现。

③ 良好的性能。

S3 是针对互联网的一种存储服务，因此它的数据访问速度不能与本地硬盘的文件访问相比。但是，从同一区域内的 Amazon EC2 可以快速访问 Amazon S3。如果同时使用多个线程、多个应用程序或多个客户端访问 S3，那 S3 累计总吞吐量往往远远超出单个服务器可以生成或消耗的吞吐量。S3 在设计上能够保证服务端的访问延时与互联网的延时相比要小很多。

为了加快相关数据的访问速度，许多开发人员将 Amazon S3 与 Amazon DynamoDB 或 Amazon RDS 配合使用。由 S3 存储实际信息，而 DynamoDB 或 RDS 则充当关联元数据（如存储对象名称、大小、关键字等）的存储。数据库提供索引和搜索的功能，而通过元数据搜索高效地找出存储对象的引用信息。然后，用户可以借助该结果准确定位存储对象本身并从 S3 中获取它。当然，为提高最终用户访问 S3 中数据的性能，还可以使用 Amazon

CloudFront 这样的 CDN 服务。

④ 接口简单。

Amazon S3 提供基于 SOAP 和 REST 两种形式的 Web 服务 API 来用于数据的管理操作。这些 API 所提供的管理和操作既针对存储桶也针对存储对象。虽然直接使用基于 SOAP 或 REST 的 API 非常灵活,但是由于这些 API 相对比较底层,因此实际使用起来相当烦琐。因此,为方便开发人员使用 AWS,专门基于 REST API 为常见的开发语言提供了高级工具包或软件开发包(SDK)。这些 SDK 支持的语言包括 Java、.NET、PHP、Ruby 和 Python 等。另外,如果需要在操作系统中直接管理和操作 S3,那么 AWS 也为 Windows 和 Linux 环境提供了一个集成的 AWS 命令行接口(CLI)。在这个命令行环境中你可以使用类似 Linux 的命令来实现常用的操作,如 ls、cp、mv、sync 等。最后,还可以通过 AWS 的 Web 管理控制台来简单地使用 S3 服务,包括创建存储桶、上传和下载数据对象等操作。当然现在也有很多第三方的工作能够帮助用户通过图形化的界面使用 S3 服务,比如 S3 Organizer (Firefox 的一个免费插件)、CloudBerry Explorer for Amazon S3 等。

2) OpenStack Swift

作为 AWS S3 的开源实现,OpenStack Swift 的出现逐渐打破了 S3 的垄断地位,它提供了弹性可伸缩、高可用的分布式对象存储服务,适合存储大规模非结构化数据。我们从 Swift 的背景、数据模型和系统架构入手进行介绍。

(1) OpenStack Swift 背景与概览

Swift 最初是由 Rackspace 公司开发的高可用分布式对象存储服务,并于 2010 年贡献给 OpenStack 开源社区作为其最初的核心子项目之一,为其 Nova 子项目提供虚拟镜像存储服务。Swift 构筑在比较便宜的标准硬件存储基础设施之上,无须采用 RAID (磁盘冗余阵列),通过在软件层面引入一致性散列技术和数据冗余性,牺牲一定程度的数据一致性来达到高可用性和可伸缩性,支持多租户模式、容器和对对象读写操作,适合解决互联网的应用场景下非结构化数据存储问题。

Swift 项目是基于 Python 开发的,采用 Apache 2.0 许可协议,可用来开发商用系统。

(2) 数据模型

Swift 采用层次数据模型,共设三层逻辑结构: Account/Container/Object (即账户/容器/对象),每层节点数均没有限制,可以任意扩展。这里的账户和个人账户不是一个概念,可理解为租户,用来做顶层的隔离机制,可以被多个个人账户所共同使用;容器代表封装一组对象,类似文件夹或目录;叶子节点代表对象,由元数据和内容两部分组成,如图 4.7 所示。

(3) Swift 系统架构

Swift 采用完全对称、面向资源的分布式系统架构设计,所有组件都可扩展,避免因单点失效而扩散并影响整个系统运转;通信方式采用非阻塞式 I/O 模式,提高了系统吞吐和响应能力。

Swift 的系统架构如图 4.8 所示。

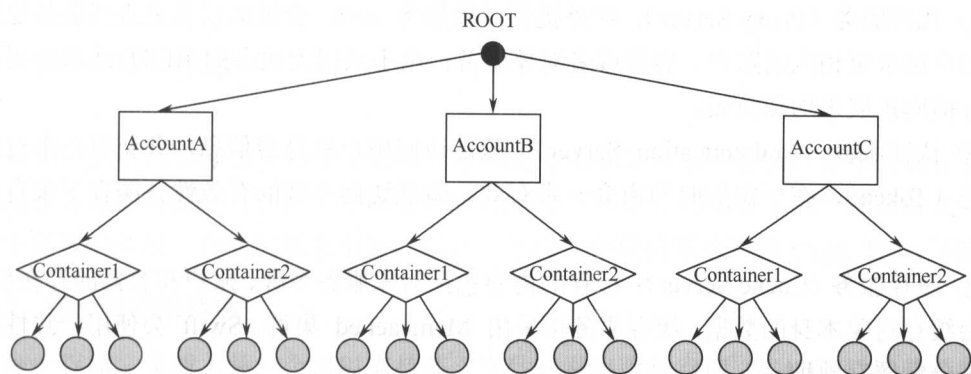


图 4.7 Swift 数据模型

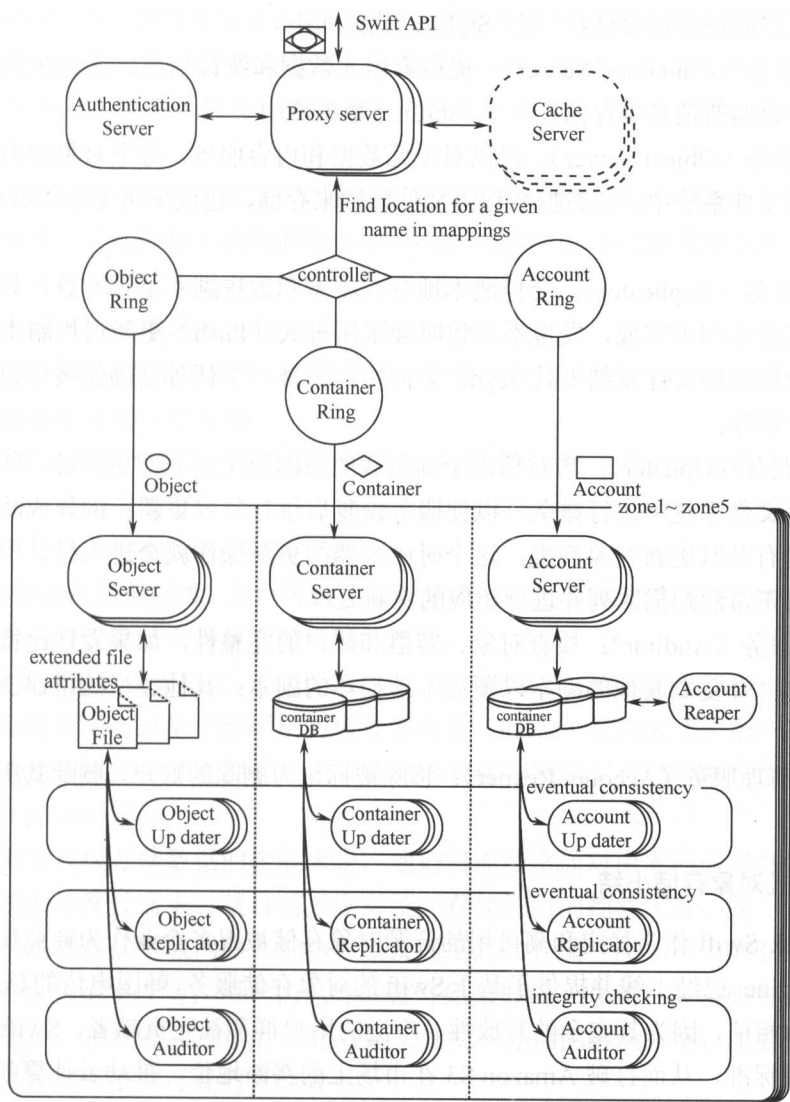


图 4.8 Swift 系统架构图

① 代理服务 (Proxy Server): 对外提供对象服务 API, 会根据信息来查找服务地址并转发用户请求至相应的账户、容器或者对象服务; 由于采用无状态的 REST 请求协议, 可以进行横向扩展来均衡负载。

② 认证服务 (Authentication Server): 验证访问用户的身份信息, 并获得一个对象访问令牌 (Token), 在一定的时间内会一直有效, 验证访问令牌的有效性并缓存下来直至过期。

③ 缓存服务 (Cache Server): 缓存的内容包括对象服务令牌、账户和容器的存在信息, 但不会缓存对象本身的数据; 缓存服务可采用 Memcached 集群, Swift 会使用一致性散列算法来分配缓存地址。

④ 账户服务 (Account Server): 提供账户元数据和统计信息, 并维护所含容器列表的服务, 每个账户的信息被存储在一个 SQLite 数据库中。

⑤ 容器服务 (Container Server): 提供容器元数据和统计信息, 并维护所含对象列表的服务, 每个容器的信息也存储在一个 SQLite 数据库中。

⑥ 对象服务 (Object Server): 提供对象元数据和内容服务, 每个对象的内容会以文件的形式存储在文件系统中, 元数据会作为文件属性来存储, 建议采用支持扩展属性的 XFS 文件系统。

⑦ 复制服务 (Replicator): 会检测本地分区副本和远程副本是否一致, 具体通过对比散列文件和高级水印来完成, 发现不一致时会采用推式 (Push) 更新远程副本, 例如对象复制服务会使用远程文件复制工具 rsync 来同步; 另外一个任务是确保被标记删除的对象从文件系统中移除。

⑧ 更新服务 (Updater): 当对象由于高负载的原因而无法立即更新时, 任务将会被序列化到在本地文件系统中进行排队, 以便服务恢复后进行异步更新; 例如成功创建对象后容器服务器没有及时更新对象列表, 这个时候容器的更新操作就会进入排队中, 更新服务会在系统恢复正常后扫描队列并进行相应的更新处理。

⑨ 审计服务 (Auditor): 检查对象、容器和账户的完整性, 如果发现比特级的错误, 文件将被隔离, 并复制其他的副本以覆盖本地损坏的副本; 其他类型的错误会被记录到日志中。

⑩ 账户清理服务 (Account Reaper): 移除被标记为删除的账户, 删除其所包含的所有容器和对象。

3. 分布式对象存储小结

OpenStack Swift 作为稳定和高可用的开源对象存储被很多企业作为商业化部署, 如新浪的 App Engine 已经上线并提供了基于 Swift 的对象存储服务, 韩国电信的 Ucloud Storage 服务。有理由相信, 因为其完全的开放性、广泛的用户群和社区贡献者, Swift 可能会成为云存储的开放标准, 从而打破 Amazon S3 在市场上的垄断地位, 推动云计算朝着更加开放和可互操作的方向前进。

4.3.4 统一存储

前面讨论了云存储系统的3个分类，分别是分布式文件系统存储、分布式块存储和分布式对象存储。所谓统一存储，可以说是同时支持以上3种存储技术的一种集成式解决方案。下面我们从统一存储的概念出发，结合一个统一存储的系统实例 Ceph 来进行描述。

1. 统一存储的概念

统一存储，实质上是一个可以支持基于文件的网络附加存储（NAS）以及基于数据块的 SAN 的网络化的存储架构。由于其支持不同的存储协议为主机系统提供数据存储，因此也被称为多协议存储，这些多协议系统可以通过网络接口或者光纤通道连接到服务器上。

统一存储概念的出现，需要追溯到十余年之前。在过去的十年里，统一存储发展的势头却一直不温不火。但发展至最近两年，统一存储开始迸发出新的能量，重新成为了存储厂商之间的夺金点。

关于统一存储的定义，简而言之，就是既支持基于文件的 NAS 存储，包括 CIFS、NFS 等文件协议类型，又支持基于块数据的 SAN 存储，包括 FC、iSCSI 等访问协议，并且可由一个统一界面进行管理。

在数据存储架构中部署统一存储系统有着如下优势。

① 规划整体存储容量的能力：通过部署一个统一存储系统可以省去对文件存储容量以及数据块存储容量分别进行规划。

② 利用率可以得到提升，容量本身并没有标准限制：统一存储可以避免与分别对数据块及文件存储支持相关的容量利用率方面的问题，用户不必担心买多了支持其中一种协议而少买了支持另外一种协议的存储。

③ 存储资源池的灵活性：用户可以在无须知道应用是否需要数据块或者文件数据访问的情况下，自由分配存储来满足应用环境的需要。

④ 积极支持服务器虚拟化：在很多时候，用户在部署他们的服务器虚拟化环境时都会因为性能方面的要求而对基于数据块的裸设备映射（RDM）提出要求。统一存储为用户如何存储他们的虚拟机提供了选择，而无须像之前那样分别购买存储区域网络（SAN）和网络附件存储（NAS）设备。

对于需要大规模存储数据的企业来说，他们可能经常面对由应用的特殊性能要求而带来的存储系统的特殊性需求。就目前大家对统一存储使用的趋势来看，统一存储将会在许多次级应用上取代存储区域网络（SAN）以及网络附加存储（NAS）。2008 年企业战略集团（ESG）的研究结果表明，用户们已经陆续开始采取统一存储，参与研究调查的人群中有 70% 的人表示，他们在实施或者计划开始实施统一存储的解决方案，主要的驱动力在于统一存储解决方案能带来可观的存储效率。

2. 统一存储系统实例

统一存储的一个代表实例就是 Ceph。Ceph 是开源实现的 PB 级分布式文件系统，其分布式对象存储机制为上层提供了文件接口、块存储接口和对象存储接口。我们从 Ceph 的基本概念入手，分析其特点、设计目标，以及它的设计架构与组件，并着重对 Ceph 的数据分布算法做重点的介绍。

1) Ceph 的设计目标与特点

Ceph 最初是一项关于存储系统的 PhD 研究项目，由 Sage Weil 在 University of California, Santa Cruz (UCSC) 实施。

首先我们先介绍 Ceph 系统的设计目标，要知道，设计一个分布式文件系统需要多方面的努力，Ceph 的目标可以简单定义为 3 个方面：

- ① 可轻松扩展到数 PB 容量；
- ② 高可靠性；
- ③ 对多种工作负载的高性能（每秒输入/输出操作和带宽）。

对应上面所说的 Ceph 的设计目标，相对于其他分布式文件系统，Ceph 统一存储文件系统有以下几个设计目标。

(1) 可扩展性

Ceph 的可扩充性主要体现在以下 3 个方面：

- ① Ceph 系统在 LGPL 许可下基于 POSIX 规范编写，具有良好的二次开发和移植性。
- ② 存储节点容量可以很容易扩展到 PB 级。
- ③ Ceph 是一个比较通用的文件系统，不像 GFS 针对大文件的场合比较适合，其对大部分 workloads 比较适合。

(2) 性能和可靠性

可扩展性和性能之间必然有一种平衡，Ceph 的存储节点的扩展带来性能降低呈现的是一种非线性的降低。而其可靠性也和目前的分布式系统一样采用 N-way 副本策略。但它也有自己的特点，即元数据不采用单一节点方式，而采用集群方式，对热点节点的元数据同样也采用了多副本策略。

(3) 负载均衡

负载均衡策略主要体现在元数据和存储节点上。元数据集群中，热点节点的元数据会迁移到新增元数据节点上。而对于存储节点，同样会定期迁移到新增节点上，从而保证所有节点的负载均衡。

2) Ceph 的系统架构与组件

Ceph 系统架构可以大致划分为四部分：客户端（数据用户）、元数据服务器（缓存和同步分布式元数据）、一个对象存储集群（将数据和元数据作为对象存储，执行其他关键职能），以及集群监视器（执行监视功能）。系统概念架构如图 4.9 所示。

Ceph 和传统的文件系统之间的重要差异之一就是，它将智能都用在了生态环境而不是

文件系统上。图 4.10 显示了一个简单的 Ceph 生态系统。Ceph client 是 Ceph 文件系统的用户。Ceph metadata daemon 提供了元数据服务器，而 Ceph object storage daemon 提供了实际存储（对数据和元数据两者）。最后，Ceph monitor 提供了集群管理。要注意的是，Ceph 客户、对象存储端点、元数据服务器（根据文件系统的容量）可以有許多，而且至少有一对冗余的监视器。

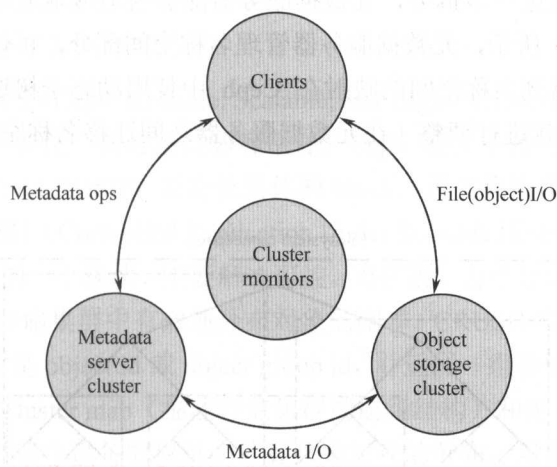


图 4.9 Ceph 概念架构

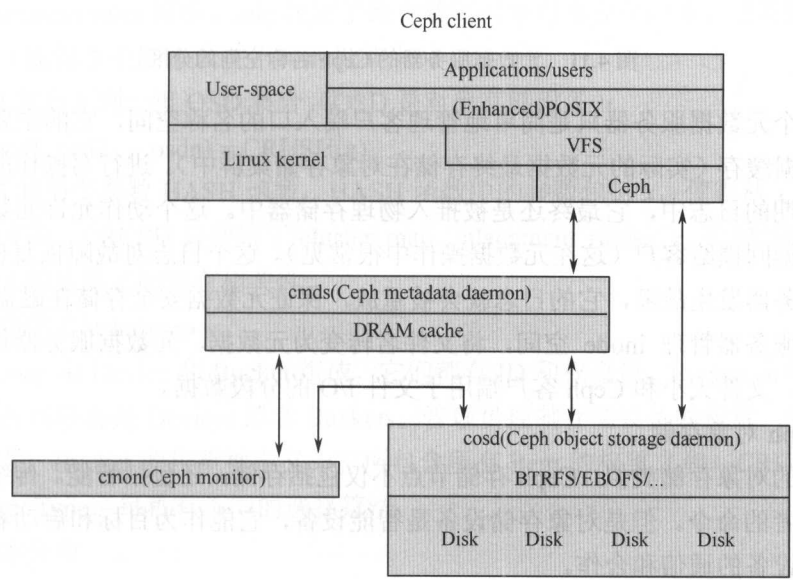


图 4.10 Ceph 简单生态系统架构

(1) Ceph 客户端

Ceph 文件系统或者至少是客户端接口是在 Linux 内核中实现的。值得注意的是，在大

多数文件系统中，所有的控制和智能在内核的文件系统源本身中执行。但是，在 Ceph 中，文件系统的智能分布在节点上，这简化了客户端接口，并为 Ceph 提供了大规模扩展能力。

（2）Ceph 元数据服务器

Ceph 元数据服务器（mdds）的工作就是管理文件系统的名称空间。虽然元数据和数据两者都存储在对象存储集群中，但两者分别管理，支持可扩展性。事实上，元数据在一个元数据服务器集群上被进一步拆分，元数据服务器能够自适应地复制和分配名称空间，避免出现热点。如图 4.11 所示，元数据服务器管理名称空间部分，可以（为冗余和性能）进行重叠。元数据服务器到名称空间的映射在 Ceph 中使用动态子树逻辑分区执行，它允许 Ceph 对变化的工作负载进行调整（在元数据服务器之间迁移名称空间），同时保留性能的位置。

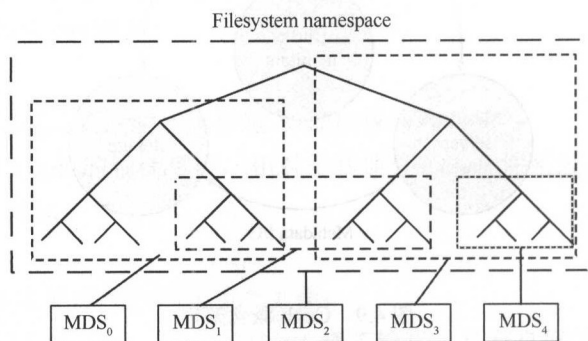


图 4.11 元数据服务器的 Ceph 名称空间的分区

因为每个元数据服务器只是简单地管理客户端入口的名称空间，它的主要应用就是一个智能元数据缓存（实际的元数据最终存储在对象存储集群中）。进行写操作的元数据被缓存存在一个短期的日志中，它最终还是被推入物理存储器中。这个动作允许元数据服务器将最近的元数据回馈给客户（这在元数据操作中很常见）。这个日志对故障恢复也很有用：如果元数据服务器发生故障，它的日志就会被重放，保证元数据安全存储在磁盘上。

元数据服务器管理 inode 空间，将文件名转变为元数据。元数据服务器将文件名转变为索引节点，文件大小和 Ceph 客户端用于文件 I/O 的分段数据。

（3）Ceph 对象存储

和传统的对象存储类似，Ceph 存储节点不仅包括存储，还包括智能。传统的驱动只响应来自启动者的命令。但是对象存储设备是智能设备，它能作为目标和启动者，支持与其他对象存储设备的通信和合作。

从存储角度来看，Ceph 对象存储设备执行从对象到块的映射（在客户端的文件系统层中常常执行的任务）。这个动作允许本地实体以最佳方式决定怎样存储一个对象。Ceph 的早期版本在一个名为 EBOFS 的本地存储器上实现一个自定义低级文件系统。这个系统实现一个到底层存储的非标准接口，这个底层存储已针对对象语义和其他特性（例如对磁盘提

交的异步通知)调优。

(4) Ceph 监视器

Ceph 包含实施集群映射管理的监视器,但是故障管理的一些要素是在对象存储本身中执行的。当对象存储设备发生故障或者添加新设备时,监视器就检测和维护一个有效的集群映射。这个功能按一种分布的方式执行,这种方式中映射升级可以和当前的流量通信。Ceph 使用 1Paxos,它是一系列分布式共识算法。

3) Ceph 的数据分布算法解析

从 Ceph 的原始论文《Ceph: Reliable, Scalable, and High-Performance Distributed Storage》来看,Ceph 专注于扩展性、高可用性和容错性。Ceph 放弃了传统的 Metadata 查表方式(HDFS),而改用算法(CRUSH)去定位具体的 block,现在我们详细剖析一下 Ceph 的数据分布算法——CRUSH (Controlled Replication Under Scalable Hashing) 算法。

CRUSH 是 Ceph 的一个模块,主要解决可控、可扩展、去中心化的数据副本分布问题。它能够在层级结构的存储集群中有效地分布对象的副本。CRUSH 实现了一种伪随机(确定性)的函数,它的参数是 object id 或 object group id,并返回一组存储设备(用于保存 object 副本)。CRUSH 需要 Cluster map (描述存储集群的层级结构)和副本分布策略(rule)。

CRUSH 算法通过每个设备的权重来计算数据对象的分布。对象分布是由 Cluster map 和 data distribution policy 决定的。Cluster map 描述了可用存储资源和层级结构(比如有多少个机架,每个机架上有多少个服务器,每个服务器上有多少个磁盘)。data distribution policy 由 placement rules 组成。rule 决定了每个数据对象有多少个副本,以及这些副本存储的限制条件(比如 3 个副本放在不同的机架中)。

CRUSH 算出 x 到一组 OSD 集合(OSD 是对象存储设备):

$(osd0, osd1, osd2 \dots osdn) = CRUSH(x)$

CRUSH 利用多参数 HASH 函数,HASH 函数中的参数包括 x ,使得从 x 到 OSD 集合是确定和独立的。CRUSH 只使用了 cluster map、placement rules、 x 。CRUSH 是伪随机算法,相似输入的结果之间没有相关性。

(1) 层级的 Cluster map

Cluster map 由 Device 和 Bucket 组成,它们都有 ID 和权重值。Bucket 可以包含任意数量 Item。Item 可以都是 Devices 或者 Buckets。管理员控制存储设备的权重。权重和存储设备的容量有关。Bucket 的权重被定义为它所包含所有 item 的权重之和。CRUSH 基于 4 种不同的 Bucket Type,每种有不同的选择算法。

(2) 副本分布

副本在存储设备上的分布影响数据的安全。Cluster map 反映了存储系统的物理结构。CRUSH placement policies 决定把对象副本分布在不同的区域(某个区域发生故障时并不会影响其他区域)。每个 rule 包含一系列操作(用在层级结构上)。

这些操作如下。

① $tack(a)$: 选择一个 Item, 一般是 Bucket, 并返回 Bucket 所包含的所有 Item。这些 Item 是后续操作的参数, 这些 Item 组成向量 i 。

② $select(n, t)$: 迭代操作每个 Item (向量 i 中的 Item), 对于每个 Item (向量 i 中的 Item) 向下遍历 (遍历这个 Item 所包含的 Item), 都返回 n 个不同的 Item (Type 为 t 的 Item), 并把这些 Item 都放到向量 i 中。 $select$ 函数会调用 $c(r, x)$ 函数, 这个函数会在每个 Bucket 中伪随机选择一个 Item。

③ $emit$: 把向量 i 放到 $result$ 中。

存储设备有一个确定的类型。每个 Bucket 都有 Type 属性值, 用于区分不同的 Bucket 类型 (比如 row、rack、host 等, Type 可以自定义)。rules 可以包含多个 take 和 emit 语句块, 这样就允许从不同的存储池中选择副本的 storage target。

(3) 冲突、故障、超载

$select(n, t)$ 操作会循环选择第 $r=1, \dots, n$ 个副本, r 作为选择参数。在这个过程中, 假如选择到的 Item 遇到 3 种情况 (冲突、故障、超载) 时, CRUSH 会拒绝选择这个 Item, 并使用 r' (r' 和 r 、出错次数、firstn 参数有关) 作为选择参数重新选择 Item。

① 冲突: 这个 Item 已经在向量 i 中, 已被选择。

② 故障: 设备发生故障, 不能被选择。

③ 超载: 设备使用容量超过警戒线, 没有剩余空间保存数据对象。

故障设备和超载设备会在 cluster map 上标记 (还留在系统中), 这样避免了不必要的数据迁移。

(4) MAP 改变和数据迁移

当添加移除存储设备, 或有存储设备发生故障时 (cluster map 发生改变时), 存储系统中的数据会发生迁移。好的数据分布算法可以最小化数据迁移大小。

(5) Bucket 的类型

CRUSH 映射算法解决了效率和扩展性这两个矛盾的目标。而且当存储集群发生变化时, 可以最小化数据迁移, 并重新恢复平衡分布。CRUSH 定义了四种具有不同算法的 Buckets。每种 Bucket 基于不同的数据结构, 并有不同的 $c(r, x)$ 伪随机选择函数。

不同的 Bucket 有不同的性能和特性。

① Uniform Buckets: 适用于具有相同权重的 Item, 而且 Bucket 很少添加删除 Item。它的查找速度是最快的。

② List Buckets: 它的结构是链表结构, 所包含的 Item 可以具有任意的权重。CRUSH 从表头开始查找副本的位置, 它先得到表头 Item 的权重 W_h 、剩余链表中所有 Item 的权重之和 W_s , 然后根据 $hash(x, r, item)$ 得到一个 $[0 \sim 1]$ 的值 v , 假如这个值 v 在 $[0 \sim W_h/W_s]$ 之中, 则副本在表头 Item 中, 并返回表头 Item 的 ID。否则继续遍历剩余的链表。

③ Tree Buckets: 其结构如图 4.12 所示, 链表的查找复杂度是 $O(n)$, 决策树的查找复杂度是 $O(\log n)$ 。Item 是决策树的叶子节点, 决策树中的其他节点知道它左右子树的权重,

节点的权重等于左右子树的权重之和。CRUSH 从 root 节点开始查找副本的位置，它先得到节点的左子树的权重 W_l ，得到节点的权重 W_n ，然后根据 $\text{hash}(x, r, \text{node_id})$ 得到一个 $[0 \sim 1]$ 的值 v ，假如这个值 v 在 $[0 \sim W_l/W_n)$ 中，则副本在左子树中，否则在右子树中。继续遍历节点，直到到达叶子节点。Tree Bucket 的关键是当添加删除叶子节点时，决策树中的其他节点的 node_id 不变。决策树中节点的 node_id 的标识是根据对二叉树的中序遍历来决定的 (node_id 不等于 Item 的 ID，也不等于节点的权重)。

④ Straw Buckets: 这种类型让 Bucket 所包含的所有 Item 公平地竞争 (不像 list 和 tree 一样需要遍历)。这种算法就像抽签一样，所有的 Item 都有机会被抽中 (只有最长的签才能被抽中)。每个签的长度是由 $\text{length} = f(W_i) * \text{hash}(x, r, i)$ 决定的， $f(W_i)$ 和 item 的权重有关， i 是 item 的 id 号。 $c(r, x) = \text{MAX}_i(f(W_i) * \text{hash}(x, r, i))$ 。

不同 Bucket 的算法复杂度和数据迁移大小见表 4.1。

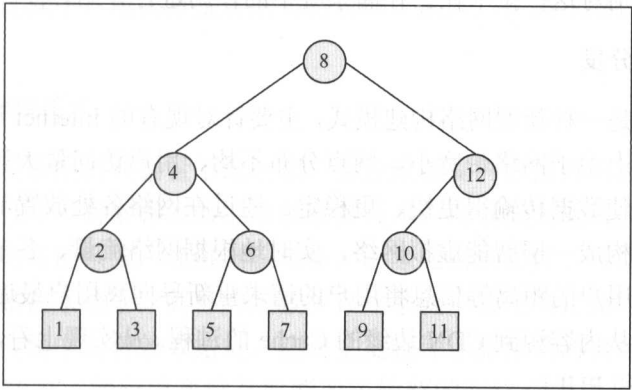


图 4.12 Tree Buckets 的结构

表 4.1 不同 Bucket 的算法复杂度和数据迁移大小

Action	Uniform	List	Tree	Straw
Speed	$O(1)$	$O(n)$	$O(\log n)$	$O(n)$
Additions	poor	optimal	good	optimal
Removals	poor	poor	good	optimal

3. 统一存储小结

在实施统一存储系统的过程中所面临的问题之一是有关组织惯性的。一般来说，存储区域网络 (SAN) 和网络附加存储 (NAS) 管理在不同的组里，通常 SAN 由数据存储管理员管理，而 NAS 由系统或者网络管理员管理。容量划分也通常分开执行，因为他们对共享资源感到恐惧，并且害怕划分容量可能会导致性能问题。此外，他们认为在管理文件共享资源和磁盘容量之间的细微差距上也存在问题。有这些顾虑都是正常的，并且需要知道的是，加固共享平台对降低需要被管理的文件共享没有太多帮助，这本身也是一个挑战。然

而这是一个很简单的道理，共享资源意味着将会使用更少的系统（更低的运营成本）以及灵活的可以合并的存储容量（对业务的响应能力）。

4.4 其他相关技术

通过前面几节的介绍，想必大家对云存储系统有了大致的了解了。除了上述讨论的云存储系统实现的几种关键技术，还有很多其他技术，为云存储系统各个方面提供保障，比如安全、性能等。那么本节把这些辅助技术挑选出来，做一个概要的介绍，读者只需要了解它们的功能。这些技术里有 CDN 内容分发、数据备份技术、数据压缩技术、重复数据删除技术、数据加密技术。

下面我们将分别对这些技术在云存储系统中的作用进行介绍。

1. CDN 内容分发

内容分发网络是一种新型网络构建模式，主要针对对现有的 Internet 进行改造。基本思想是尽量避开互联网上由于网络带宽小、网点分布不均、用户访问量大等影响数据传输速率和稳定性的弊端，使数据传输得更快、更稳定。通过在网络各处放置节点服务器，在现有互联网的基础之上构成一层智能虚拟网络，实时地根据网络流量、各节点的连接和负载情况、响应时间、到用户的距离等信息将用户的请求重新导向离用户最近的服务节点上。

内容分发包含从内容源到 CDN 边缘的 Cache 的过程。从实现上看，有两种主流的内容分发技术：PUSH 和 PULL。

PUSH 是一种主动分发的技术。通常，PUSH 由内容管理系统发起，将内容从源或者中心媒体资源库分发到各边缘的 Cache 节点。分发的协议可以采用 HTTP/FTP 等。通过 PUSH 分发的内容一般是比较热点的内容，这些内容通过 PUSH 方式预分发（Preload）到边缘 Cache，可以实现有针对性的内容提供。对于 PUSH 分发需要考虑的主要问题是分发策略，即在什么时候分发什么内容。一般来说，内容分发可以由 CP（内容提供商）或者 CDN 内容管理员人工确定，也可以通过智能的方式决定，即所谓的智能分发。它根据用户访问的统计信息，以及预定义的内容分发的规则，确定内容分发的过程。

PULL 是一种被动的分发技术，PULL 分发通常由用户请求驱动。当用户请求的内容在本地的边缘 Cache 上不存在（未命中）时，Cache 启动 PULL 方法从内容源或者其他 CDN 节点实时获取内容。在 PULL 方式下，内容的分发是按需的。

在实际的 CDN 系统中，一般两种分发方式都支持，但是根据内容的类型和业务模式的不同，在选择主要的内容分发方式时会有所不同。通常，PUSH 的方式适合内容访问比较集中的情况，如热点的影视流媒体内容；PULL 方式比较适合内容访问分散的情况。

在内容分发的过程中，对于 Cache 设备而言，关键是需要建立内容源 URL、内容发布的 URL、用户访问的 URL，以及内容在 Cache 中存储的位置之间的映射关系。

2. 数据备份技术

在以数据为中心的时代，数据的重要性无可置否，如何保护数据是一个永恒的话题，即便是现在的云存储发展时代，数据备份技术也非常重要。数据备份技术是将数据本身或者其中的部分在某一时间的状态以特定的格式保存下来，以备原数据出现错误、被误删除、恶意加密等各种原因不可用时，可快速准确地将数据进行恢复的技术。数据备份是容灾的基础，是为防止突发事件而采取的一种数据保护措施，根本目的是数据资源重新利用和保护，核心的工作是数据恢复。

3. 数据压缩技术

数据压缩技术，就是用最少的数码来表示信号的技术。由于云存储中的数据量特别庞大，如果不对其进行有效的压缩，不但会造成对存储硬件设备的大量占用，而且还会造成数据在网络传输的过程中占用大量宝贵的带宽资源。因此，数据压缩技术是云存储中相当关键的一个技术。

4. 重复数据删除技术

随着数据中重复数据的数据量不断增加，会导致重复的数据占用更多的空间。重复数据删除（Dedupe）技术一种非常高级的数据缩减技术，可以极大地减少备份数据的数量，通常用于基于磁盘的备份系统，通过删除运算，消除冗余的文件、数据块或字节，以保证只有单一的数据存储在系统中。其目的是减少存储系统中使用的存储容量，增大可用的存储空间，增加网络传输中的有效数据量。然而重复删除运算相当消耗运算资源，对存取能效会造成相当程度冲击，如应用在对存取能效较敏感的网络存储设备上，将会面临许多困难。

存储系统的重复数据删除过程一般是这样的：首先将数据文件分割成一组数据块，为每个数据块计算指纹，然后以指纹为关键字进行 HASH 查找，匹配则表示该数据块为重复数据块，仅存储数据块索引号，否则则表示该数据块是一个新的唯一块，对数据块进行存储并创建相关元信息。这样，一个物理文件在存储系统就对应一个逻辑表示——由一组 FP 组成的元数据。当读取文件时，先读取逻辑文件，然后根据 FP 序列，从存储系统中取出相应数据块，还原物理文件副本。从如上过程中可以看出，Dedupe 的关键技术主要包括文件数据块切分、数据块指纹计算和数据块检索。

（1）文件数据块切分

Dedupe 按照消重的粒度可以分为文件级和数据块级。文件级的 Dedupe 技术也称单一实例存储（Single Instance Store, SIS），数据块级的重复数据删除的消重粒度更小，可以达到 4~24KB。显然，数据块级的可以提供更高的数据消重率，因此目前主流的 Dedupe 产品都是数据块级的。数据分块算法主要有三种，即定长切分（fixed-size partition）、CDC 切分（content-defined chunking）和滑动块（sliding block）切分。定长分块算法采用预先定义好的块大小对文件进行切分，并进行弱校验和 MD5 强校验。弱校验主要是为了提升差异编

码的性能,先计算弱校验值并进行 HASH 查找,如果发现则计算 MD5 强校验值并进行进一步 HASH 查找。由于弱校验值计算量要比 MD5 小很多,因此可以有效提高编码性能。定长分块算法的优点是简单、性能高,但它对数据插入和删除非常敏感,处理十分低效,不能根据内容变化做调整和优化。

CDC (content-defined chunking) 算法是一种变长分块算法,它应用数据指纹(如 Rabin 指纹)将文件分割成长度大小不等的分块策略。与定长分块算法不同,它是基于文件内容进行数据块切分的,因此数据块大小是可变化的。算法执行过程中,CDC 使用一个固定大小(如 48 字节)的滑动窗口对文件数据计算数据指纹。如果指纹满足某个条件,如当它的值模特定的整数等于预先设定的数时,则把窗口位置作为块的边界。CDC 算法可能会出现病态现象,即指纹条件不能满足,块边界不能确定,导致数据块过大。实现中可以对数据块的大小进行限定,设定上下限,解决这种问题。CDC 算法对文件内容变化不敏感,插入或删除数据只会影响到较少的数据块,其余数据块不受影响。CDC 算法也是有缺陷的,数据块大小的确定比较困难,粒度太细则开销太大,粒度过粗则效果不佳。如何在两者之间权衡折中,这是一个难点。

滑动块(sliding block)算法结合了定长切分和 CDC 切分的优点,块大小固定。它对定长数据块先计算弱校验值,如果匹配则再计算 MD5 强校验值,两者都匹配则认为是一个数据块边界。该数据块前面的数据碎片也是一个数据块,它是不定长的。如果滑动窗口移过一个块大小的距离仍无法匹配,则也认定为一个数据块边界。滑动块算法对插入和删除问题的处理非常高效,并且能够检测到比 CDC 更多的冗余数据,它的不足是容易产生数据碎片。

(2) 数据指纹计算

数据指纹是数据块的本质特征,理想状态是每个唯一数据块具有唯一的数据指纹,不同的数据块具有不同的数据指纹。数据块本身往往较大,因此数据指纹的目标是期望以较小的数据表示(如 16、32、64、128 字节)来区别不同数据块。数据指纹通常是对数据块内容进行相关数学运算获得的,从当前研究成果来看 HASH 函数比较接近于理想目标,比如 MD5、SHA1、SHA-256、SHA-512、one-Way、RabinHash 等。另外,还有许多字符串 HASH 函数也可以用来计算数据块指纹。然而,遗憾的是这些指纹函数都存在碰撞问题,即不同数据块可能会产生相同的数据指纹。相对来说,MD5 和 SHA 系列 HASH 函数具有非常低的碰撞发生概率,因此通常被采用作为指纹计算方法。其中,MD5 和 SHA1 是 128 位的,SHA-X(X 表示位数)具有更低的碰撞发生概率,但同时计算量也会大大增加。实际应用中,需要在性能和数据安全性方面做出权衡。另外,还可以同时使用多种 HASH 算法来为数据块计算指纹。

(3) 数据块检索

对于大存储容量的 Dedupe 系统来说,数据块数量非常庞大,尤其是数据块粒度细的情况下。因此,在这样一个大的数据指纹库中检索,性能就会成为瓶颈。信息检索方法有很多种,如动态数组、数据库、RB/B/B+/B*树、Hashtable 等。HASH 查找因为其 $O(1)$ 的查找

性能而著称，被对查找性能要求高的应用所广泛采用，Dedupe 技术中也采用它。Hashtable 处于内存中，会消耗大量内存资源，在设计 Dedupe 前需要对内存需求做合理规划。根据数据块指纹长度、数据块数量（可以由存储容量和平均数据块大小估算）可以估算出内存需求量。

散列表（Hashtable，也叫哈希表），是根据关键码值（Key value）而直接进行访问的数据结构。也就是说，它通过把关键码值映射到表中一个位置来访问记录，以加快查找的速度。这个映射函数叫做散列函数，存放记录的数组叫做散列表。散列表的查找过程基本上和造表过程相同，一些关键码可通过散列函数转换的地址直接找到，另一些关键码在散列函数得到的地址上产生了冲突，需要按处理冲突的方法进行查找。

5. 数据加密技术

存储加密是指当数据从前端服务器输出，或在写进存储设备之前通过系统为数据加密，以保证存放在存储设备上的数据只有授权用户才能读取。目前云存储中常用的存储加密技术有以下几种：全盘加密，全部存储数据都是以密文形式书写的；虚拟磁盘加密，存放数据之前建立加密的磁盘空间，并通过加密磁盘空间对数据进行加密；卷加密，所有用户和系统文件都被加密；文件/目录加密，对单个的文件或者目录进行加密。

4.5 练习题

1. 横向比较各大分布式云存储技术，分析其具体应用场景。
2. 对文中提到的云存储系统的分类做一个对比总结，分析各个存储技术的优缺点。
3. 课程设计：设计一个网络云盘系统，由客户端、运营管理平台和分布式文件存储集群组成。用户通过客户端进行注册、登录、注销等操作。用户可以浏览已经上传的文件，还可以进行文件上传、下载、删除等操作。

参考文献

- [1] 刘亮. 基于虚拟化与分布式技术的云存储研究[J]. 电脑知识与技术, 2012, 8(11).
- [2] Weil S A, Brandt S A, Miller E L, et al. Ceph: A scalable, high-performance distributed file system[C]//Proceedings of the 7th symposium on Operating systems design and implementation. USENIX Association, 2006: 307-320.
- [3] Weil S A. Ceph: Reliable, scalable, and high-performance distributed storage[D]. UNIVERSITY OF CALIFORNIA SANTA CRUZ, 2007.

- [4] 战科宇, 李小勇, 刘海涛. 分布式文件系统元数据服务器高可用性设计[J]. 小型微型计算机系统, 2013, 34(4).
- [5] 亚马逊 S3 服务介绍, <http://blog.csdn.net/awschina/article/details/15502205>
- [6] Ghemawat S, Gobioff H, Leung S T. The Google file system[C]//ACM SIGOPS Operating Systems Review. ACM, 2003, 37(5): 29-43.
- [7] <http://docs.openstack.org/developer/swift/>
- [8] <http://aws.amazon.com/cn/s3/>
- [9] http://www.ibm.com/developerworks/cn/cloud/library/1310_zhanghua_openstackswift/
- [10] 冬瓜头. 大话存储 2: 存储系统架构与底层原理极限剖析. 北京: 清华大学出版社, 2011.
- [11] Openstack 中国. <http://www.openstack.cn/>
- [12] Google 三大论文中文版. <http://wenku.baidu.com/view/0b383673f242336c1eb95e15.html>
- [13] 郭御风, 李琼, 刘光明, 等. 虚拟存储技术研究[J]. 计算机应用研究, 2004, 21(2): 56-57.
- [14] 重复数据删除技术研究[OL]. <http://blog.csdn.net/liuaigui/article/details/5829083>

第5章

数据采集系统

海量数据是企业大数据战略建设的基础，因此数据采集就成了大数据分析的前站。采集是大数据价值挖掘重要的一环，其后的分析挖掘都建立在采集的基础上。虽然大数据技术的意义不在于掌握规模庞大的数据信息，而在于对这些数据进行智能处理，从中分析和挖掘出有价值的信息，但前提是拥有大量的数据。绝大多数的企业现在还很难判断，到底哪些数据未来将成为资产，通过什么方式将数据提炼为现实收入。对于这一点即便是大数据服务企业也很难给出确定的答案。但有一点是肯定的，大数据时代，谁掌握了足够的数据，谁就有可能掌握未来，现在的数据采集就是将来的资产积累。

数据的采集有基于物联网传感器的采集，也有基于网络信息的数据采集。比如在智能交通中，数据的采集有基于 GPS 的定位信息采集、基于交通摄像头的视频采集、基于交通卡口的图像采集、基于路口的线圈信号采集等。而在互联网上的数据采集是对各类网络媒介，如搜索引擎、新闻网站、论坛、微博、博客、电商网站等的各种页面信息和用户访问信息进行采集，采集的内容主要有文本信息、URL、访问日志、日期和图片等。之后我们需要把采集到的各类数据进行清洗、过滤、去重等预处理并分类归纳存储。

数据的种类是丰富多样的，随着分布式计算平台的流行，越来越多的数据将存储和计算放到分布式平台上。现在许多公司的平台每天会产生大量的日志（一般为流式数据，如搜索引擎的 PV、查询等），处理这些日志需要特定的日志系统，一般而言，这些系统需要具有以下特征：

- ① 构建应用系统和分析系统的桥梁，并将它们之间的关联解耦；
- ② 支持近实时的在线分析系统和类似于 Hadoop 的离线分析系统；
- ③ 具有高可扩展性，即当数据量增加时，可以通过增加节点进行水平扩展。

实际生产环境中根据不同的应用环境和需求，产生了很多高效的数据采集工具，本章将着重介绍生产环境中比较热门的数据采集工具，包括 Flume、Scribe、Chukwa 和 Kafka 等。

5.1 Flume

Flume 是 Cloudera 提供的一个高可用、高可靠、分布式的海量日志采集、聚合和传输的系统，可用于从不同来源的系统中采集、汇总和传输大容量的日志数据到指定的数据存储中。Flume 支持在日志系统中定制各类数据发送方来收集数据；同时还可以提供对数据进行简单处理，写到各种定制数据接受方的能力。Flume 初始的发行版本目前被统称为 Flume OG (Original Generation)，属于 Cloudera。但随着 Flume 功能的扩展，Flume OG 代码工程臃肿、核心组件设计不合理、核心配置不标准等缺点暴露出来，尤其是在 Flume OG 的最后一个发行版本 0.94.0 中，日志传输不稳定的现象尤为严重。2011 年 10 月 22 日，Cloudera 完成了 Flume-728，对 Flume 进行了里程碑式的改动：重构核心组件、核心配置以及代码架构，重构后的版本统称 Flume NG (Next Generation)；改动的另一原因是将 Flume 纳入 Apache 旗下，Cloudera Flume 改名为 Apache Flume。本章将主要关注 Flume NG 的架构和特性。

Flume 支持的采集数据源包括 console、RPC (Thrift-RPC)、text (文件)、tail (UNIX tail)、syslog (syslog 日志系统，支持 TCP 和 UDP 两种模式)、exec (命令执行) 等，支持的数据接受方包括 console、text (文件)、DFS (HDFS 文件)、RPC (Thrift-RPC) 和 syslogTCP (TCP syslog 日志系统) 等。

5.1.1 Flume 架构

对于 Flume OG，可以说它是一个分布式日志收集系统，有 Master 概念，依赖于 ZooKeeper，Agent 用于采集数据，Agent 是 Flume 中产生数据流的地方，同时，Agent 会将产生的数据流传输到 Collector。对应地，Collector 用于对数据进行聚合，往往会产生一个更大的流。而对于 Flume NG，它摒弃了 Master、ZooKeeper、Collector 和 Web 配置台，只剩下 Source、Sink 和 Channel，此时一个 Agent 的概念包括 Source、Channel 和 Sink，完全由一个分布式系统变成了传输工具。不同机器之间的数据传输不再是 OG 那样由 Agent→Collector，而是由一个 Agent 端的 Sink 流向另一个 Agent 的 Source。Flume NG 的节点组成如图 5.1 所示。

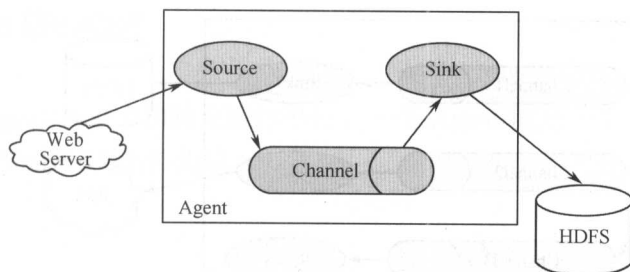


图 5.1 Flume NG 节点组成

在 Flume NG 中，有如下核心概念。

Client: 生产数据，运行在一个独立的线程上。

Source: 从 Client 收集数据，传递给 Channel。可以接收外部源发送过来的数据。不同的 Source，可以接受不同的数据格式。比如有目录池（spooling directory）数据源，可以监控指定文件夹中的新文件变化，如果目录中有文件产生，就会立刻读取其内容。

Channel: 是一个存储池，接收 Source 的输出，直到有 Sink 消费掉 Channel 中的数据。Channel 中的数据直到进入下一个 Channel 中或者进入终端才会被删除。当 Sink 写入失败后，可以自动重启，不会造成数据丢失，因此很可靠。

Sink: 消费 Channel 中的数据，然后送给外部源或者其他 Source，如数据可以写入 HDFS 或者 HBase 中。

Agent: 使用 JVM 运行 Flume。每台机器运行一个 Agent，但是可以在一个 Agent 中包含多个 Sources 和 Sinks。

Events: Flume NG 传输的数据的基本单位是 Event，如果是文本文件，通常是一行记录，这也是事务的基本单位。

Flume 的数据流由事件（Event）贯穿始终。事件是 Flume 的基本数据单位，它携带日志数据（字节数组形式）并且携带头信息，这些 Event 由 Agent 外部的 Source，比如图 5.1 中的 Web Server 生成。当 Source 捕获事件后会进行特定的格式化，然后 Source 会把事件推入（单个或多个）Channel 中。可以把 Channel 看成一个缓冲区，它将保存事件直到 Sink 处理完该事件。Sink 负责持久化日志或者把事件推向另一个 Source。值得注意的是，Flume 提供了大量内置的 Source、Channel 和 Sink 类型。不同类型的 Source、Channel 和 Sink 可以自由组合。组合方式基于用户设置的配置文件，非常灵活。比如：Channel 可以把事件暂存在内存里，也可以持久化到本地硬盘上。Sink 可以把日志写入 HDFS、HBase，甚至是另外一个 Source 等。Flume 支持用户建立多级流，也就是说，多个 Agent 可以协同工作，并且支持 Fan-in、Fan-out、Contextual Routing、Backup Routes，如图 5.2 所示。

Flume 允许多个 Agent 连在一起，形成前后相连的多级数据采集结构，如图 5.3 所示。

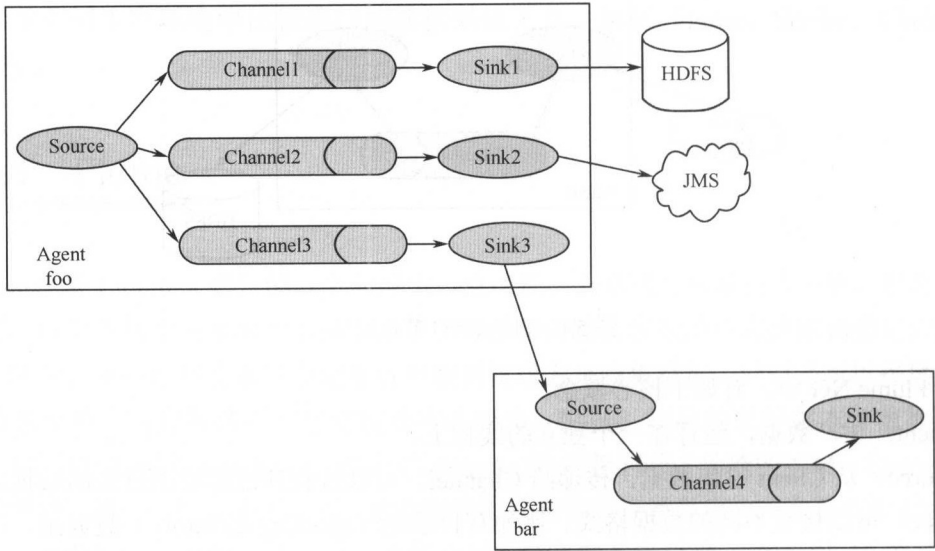


图 5.2 Flume 数据流

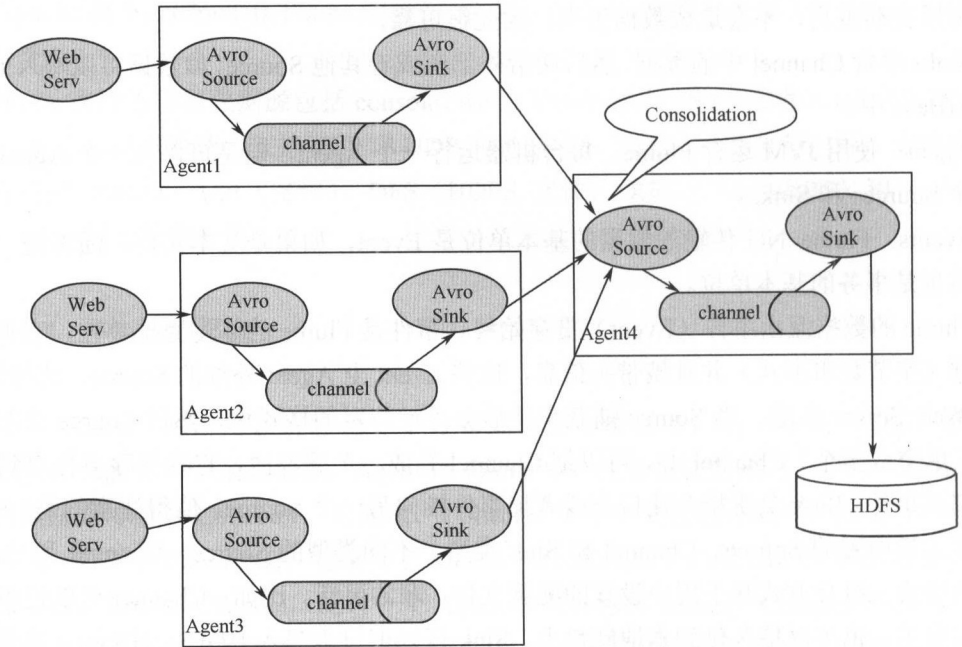


图 5.3 多级数据采集结构

5.1.2 Flume 核心组件

FlumeNG 以 Agent 为最小的独立运行单位。一个 Agent 就是一个 JVM。单 Agent 由 Source、Channel 和 Sink 三大组件构成。

1. Source

Flume 支持 Avro、log4j、syslog 和 http post (body 为 json 格式)。可以让应用程序同已有的 Source 直接打交道, 如 AvroSource、SyslogTcpSource。也可以写一个 Source, 以 IPC 或 RPC 的方式接入自己的应用, Avro 和 Thrift 都可以 (分别有 NettyAvroRpcClient 和 ThriftRpcClient 实现了 RpcClient 接口), 其中 Avro 是默认的 RPC 协议。具体代码级别的 Client 端数据接入, 可以参考官方手册。对现有程序改动最小的使用方式是直接读取程序原来记录的日志文件, 基本可以实现无缝接入, 不需要对现有程序进行任何改动。对于直接读取文件 Source, 有两种方式。

① ExecSource: 以运行 Linux 命令的方式, 持续地输出最新的数据, 如 tail -F 文件名指令, 在这种方式下, 文件名必须是指定的。ExecSource 可以实现对日志的实时收集, 但是 Flume 不运行或者指令执行出错时, 将无法收集到日志数据, 无法保证日志数据的完整性。

② SpoolSource: 监测配置的目录下新增的文件, 并将文件中的数据读取出来。需要注意两点: 复制到 spool 目录下的文件不可以再打开编辑, spool 目录下不可包含相应的子目录。SpoolSource 虽然无法实现实时收集数据, 但是可以以分钟分割文件, 趋近于实时。如果应用无法实现以分钟切割日志文件, 可以将两种收集方式结合使用。在实际使用的过程中, 可以结合 log4j 使用, 使用 log4j 的时候, 将 log4j 的文件分割机制设为 1 分钟一次, 将文件复制到 spool 的监控目录, log4j 有一个 TimeRolling 插件, 可以把 log4j 分割文件到 spool 目录, 基本实现了实时的监控。Flume 在传完文件之后, 将会修改文件的后缀, 变为.COMPLETED (后缀也可以在配置文件中灵活指定)。

2. Channel

当前有几个 Channel 可供选择, 分别是 Memory Channel、JDBC Channel、File Channel 和 Psuedo Transaction Channel 等。比较常见的是前三种 Channel。

① Memory Channel 可以实现高速的吞吐, 但是无法保证数据的完整性。

② File Channel 保证数据的完整性与一致性。在具体配置 File Channel 时, 建议 File Channel 的目录和程序日志文件保存的目录设成不同的磁盘, 以便提高效率。

File Channel 是一个持久化的隧道 (Channel), 它持久化所有的事件, 并将其存储到磁盘中。因此, 即使 Java 虚拟机宕掉, 或者操作系统崩溃或重启, 或者事件没有在管道中成功地传递到下一个代理 (Agent), 这一切都不会造成数据丢失。Memory Channel 是一个不稳定的隧道, 其原因是由于它在内存中存储所有事件。如果 Java 进程宕掉, 任何存储在内存

存的事件将会丢失。另外，内存的空间受到 RAM 大小的限制，而 File Channel 在这方面有优势，只要磁盘空间足够，它就可以将所有事件数据存储到磁盘上。

3. Sink

Sink 在设置存储数据时，可以向文件系统、数据库、Hadoop 存数据，在日志数据较少时，可以将数据存储到文件系统中，并且设定一定的时间间隔保存数据。在日志数据较多时，可以将相应的日志数据存储到 Hadoop 中，便于日后进行相应的数据分析。

5.1.3 Flume 环境搭建与部署

1. 主机资源规划

Flume 日志采集工具从结构图上可以看出，它是一个集群式的部署方式，故在选择主机时，要按照主机所担当的角色来划分主机。在 Flume 日志采集系统的系统架构中，共有三种主机角色：日志采集源（Source）、渠道/缓冲区（Channel）和目标主机（Sink）。依照目前测试环境的资源配置，划分见表 5.1～表 5.3。

表 5.1 Flume Source 服务器

Source	Hostname	IP
Source1	compute1	192.168.1.93
Source2	debian	192.168.2.17

表 5.2 Flume Channel 服务器

Channel	Hostname	IP
Channel1	compute2	192.168.1.13
Channel2	debian	192.168.2.18

表 5.3 Flume Sink 服务器

Sink	Hostname	IP
Sink1	controler	192.168.1.65

2. 环境要求

Flume 的运行环境比较简单，对操作系统基本没有特别的要求。只要保证运行环境中安装了 JDK 即可。但依据目前 Flume 的版本情况，JDK 的版本要求在 JDK 1.6 以上。另外，鉴于此次测试我们需要结合 Hadoop 提供的 HDFS 存储方式存储文件，所以我们还要求在 Sink 端安装 Hadoop 软件。

3. 安装组件

Source 端：JDK 1.7 和 Flume 1.4.0。

Channel 端: JDK 1.7 和 Flume 1.4.0。

Sink 端: JDK 1.7、Hadoop 1.2.1 和 Flume 1.4.0。

4. Flume 安装

Flume 最多使用三个组件安装, 并且这三个组件均不需要安装, 只要传到服务器上解压以后就可以使用了。所以 5 台机器的安装步骤除 Sink 端需要多安装一个 Hadoop 外, 其余 4 台机器安装步骤相同。需要的包如下:

- ① apache-flume-1.3.1-bin.tar.gz
- ② jdk-7u45-linux-x64.tar.gz
- ③ hadoop-1.2.1.tar.gz (只在 Sink 端安装即可)

默认安装路径为/usr/local 目录。

5. 配置环境变量

软件包上传完成后, 在各自的主机上要设置环境变量。编辑/etc/profile 文件, 添加如下配置:

```
export JAVA_HOME=/usr/local/jdk1.7.0_45
export JRE_HOME=/usr/local/jdk1.7.0_45/jre
export CLASSPATH=.:$JAVA_HOME/lib:$JRE_HOME/lib:$CLASSPATH
export PATH=$JAVA_HOME/bin:$JRE_HOME/bin:$PATH

export FLUME_HOME=/usr/local/apache-flume-1.4.0-bin
export FLUME_CONF_DIR=$FLUME_HOME/conf
export PATH=.:$PATH::$FLUME_HOME/bin
```

6. 配置 Flume

使用 Flume 日志采集工具的关键是对于 Flume 目录结构中 conf 目录下的*.conf 文件的配置, 此文件是对日志文件的源、传输方式、渠道和目标的配置。下面结合实验对 Flume 的配置文件进行配置。

日志采集源主机 192.168.1.93 中/usr/local/apache-flume-1.3.1-bin/conf/flume.conf 配置如下:

```
#向 agent1 采集实例中注册一个采集源名称 sc1
agent1.sources = sc1
#向 agent1 采集实例中注册一个缓存频道 ch1
agent1.channels = ch1
#向 agent1 采集实例中注册一个输出槽 sk1
agent1.sinks = sk1
#以下为 sc1 采集源的配置
```

```

#采集源类型 spooldir
agent1.sources.sc1.type = spooldir
#采集到的数据通过 ch1 缓存频道缓存
agent1.sources.sc1.channels = ch1
#采集文件夹
agent1.sources.sc1.spoolDir = /opt/tomcat/8spacelog
#已采集标识 .s
agent1.sources.sc1.fileSuffix =.s
#以下为 ch1 缓冲频道的配置
#缓冲频道类型 file
agent1.channels.ch1.type = file
#检查点文件目录
agent1.channels.ch1.checkpointDir = /var/flume/data/checkpoint
#缓存数据文件夹
agent1.channels.ch1.dataDirs = /var/flume/data
#以下为输出槽设置
#输出槽类型为 Avro
agent1.sinks.sk1.type = avro
#输出哪个缓存频道中的数据
agent1.sinks.sk1.channel = ch1
#输出到哪个主机
agent1.sinks.sk1.hostname = 192.168.1.13
#输出到哪个端口
agent1.sinks.sk1.port = 41414

```

日志缓冲主机 192.168.1.13 中的/usr/local/apache-flume-1.3.1-bin/conf/flume.conf 配置如下:

```

#向 agent1 采集实例中注册一个采集源名称 sc1
agent1.sources = sc1
#向 agent1 采集实例中注册一个缓存频道 ch1
agent1.channels = ch1
#向 agent1 采集实例中注册一个输出槽 sk1
agent1.sinks = sk1
#以下为 sc1 采集源的配置
#采集源类型 avro
agent1.sources.sc1.type = avro
#采集到的数据通过 ch1 缓存频道缓存

```

```

agent1.sources.sc1.channels = ch1
#绑定 IP 地址
agent1.sources.sc1.bind = 0.0.0.0
#监听端口号
agent1.sources.sc1.port = 41414
#以下为 ch1 缓冲频道的配置
#缓冲频道类型 file
agent1.channels.ch1.type = file
#检查点文件目录
agent1.channels.ch1.checkpointDir = /var/flume/data/checkpoint
#缓存数据文件夹
agent1.channels.ch1.dataDirs = /var/flume/data
#以下为输出槽设置
#输出槽类型为 Avro
agent1.sinks.sk1.type = avro
#输出哪个缓存频道中的数据
agent1.sinks.sk1.channel = ch1
#输出到哪个主机
agent1.sinks.sk1.hostname = 192.168.1.65
#输出到哪个端口
agent1.sinks.sk1.port = 41414

```

日志缓冲主机 192.168.1.65 中的/usr/local/apache-flume-1.3.1-bin/conf/flume.conf 配置如下:

```

#向 agent1 采集实例中注册一个采集源名称 sc1
agent1.sources = sc1
#向 agent1 采集实例中注册一个缓存频道 ch1
agent1.channels = ch1
#向 agent1 采集实例中注册一个输出槽 sk1
agent1.sinks = sk1
#以下为 sc1 采集源的配置
#采集源类型 avro
agent1.sources.sc1.type = avro
#采集到的数据通过 ch1 缓存频道缓存
agent1.sources.sc1.channels = ch1
#绑定 IP 地址

```



```

agent1.sources.sc1.bind = 0.0.0.0
#监听端口号
agent1.sources.sc1.port = 41414
agent1.sources.sc1.interceptors = i1
agent1.sources.sc1.interceptors.i1.type = timestamp
#缓冲频道类型 file
agent1.channels.ch1.type = file
#检查点文件目录
agent1.channels.ch1.checkpointDir = /var/flume/data/checkpoint
#缓存数据文件夹
agent1.channels.ch1.dataDirs = /var/flume/data
#以下为输出槽设置
#输出槽类型为 hdfs
agent1.sinks.sk1.type = hdfs
#输出哪个缓存频道中的数据
agent1.sinks.sk1.channel = ch1
#hdfs 路径按年月日进行日志文件的分文件夹保存
agent1.sinks.sk1.hdfs.path =hdfs:#192.168.1.65:9000/flume/events/%y-%m-%d
#存储文件前缀
agent1.sinks.sk1.hdfs.filePrefix = events-
#是否进行轮循归档
agent1.sinks.sk1.hdfs.round = false
#是否使用用户本地时间戳，如果设置为 false 则不能使用上面的文件夹创建方式
agent1.sinks.sk1.hdfs.useLocalTimeStamp = true

```

7. 日志采集测试

Flume 安装配置完成后，我们可以启动来测试配置完成情况。首先到两个源主机的 /opt/tomcat/8spacelog/路径下，建立 2014-11-12test.log、2014-11-13test.log、2014-11-14test.log 和 anxiang.log、anxiang1.log 日志文件，作为日志采集的文件源。

切换到 \$FLUME_HOME/conf 目录下，执行如下命令：

```
flume-ng agent -c . -f flume.conf -n agent1 -Dflume.root.logger=INFO,console---
```

这里 Flume 进程的启动顺序为：

- ① 启动 sink 主机的 flume 进程；
- ② 启动渠道主机的 flume 进程；
- ③ 启动源主机的 flume 进程。

日志采集完成后，Hadoop 会通过 HDFS 文件系统将文件存在按照日志命名的目录中，

同时可以在 Sink 主机上通过命令：

```
hadoop dfs -ls <日志存放路径>
```

查询同步的日志名称，也可以切换到/hadoop/dfs/data/current 目录中直接查看文件。

5.2 Scribe

Scribe 是 Facebook 开源的日志收集系统，在 Facebook 内部已经得到大量的应用。它能够从各种日志源上收集日志，存储到一个中央存储系统（可以是 NFS、分布式文件系统等）上，以便于进行集中统计分析处理。它为日志的“分布式收集，统一处理”提供了一个可扩展、高容错的方案。当中央存储系统的网络或者机器出现故障时，Scribe 会将日志转存到本地或者另一个位置，当中央存储系统恢复后，Scribe 会将转存的日志重新传输给中央存储系统。

5.2.1 Scribe 架构

Scribe 的架构比较简单，主要包括三部分，分别为 Scribe agent、Scribe 和存储系统（图 5.4）。

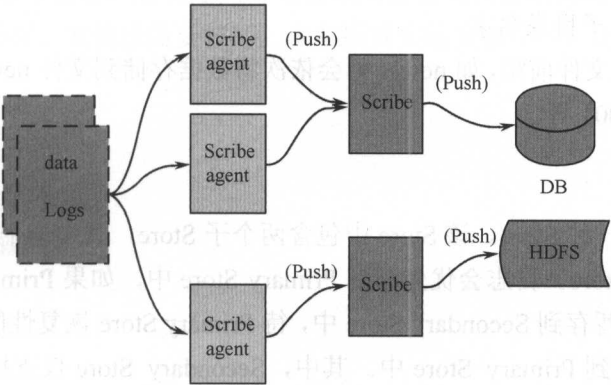


图 5.4 Scribe 架构

1. Scribe agent

Scribe agent 实际上是一个 Thrift client。向 Scribe 发送数据的唯一方法是使用 Thrift client，Scribe 内部定义了一个 Thrift 接口，用户使用该接口将数据发送给 server。

2. Scribe

Scribe 接收到 Thrift client 发送过来的数据, 根据配置文件, 将不同 topic 的数据发送给不同的对象。Scribe 提供了各种各样的 Store, 如 file、HDFS 等, Scribe 可将数据加载到这些 Store 中。

3. 存储系统

存储系统实际上就是 Scribe 中的 Store, 当前 Scribe 支持非常多的 Store, 包括 file (文件)、buffer (双层存储, 一个主储存, 一个副存储)、network (另一个 Scribe 服务器)、bucket (包含多个 Store, 通过 HASH 将数据存到不同 Store 中)、null (忽略数据)、Thriftfile (写到一个 Thrift TFileTransport 文件中) 和 multi (把数据同时存放到不同 Store 中)。

5.2.2 Scribe 中的 Store

1. file

将日志写到文件或者 NFS 中。目前支持两种文件格式, 即 std 和 HDFS, 分别表示普通文本文件和 HDFS。可配置的选项如下。

max_size: 文件大小上限, 即当文件大小达到 max_size 时, 创建新的文件继续存储数据。

rotate_period: 文件创建周期, 可以是 hourly、daily、never 和 number[suffix]。suffix 可以是 s (second), m (minute), h (hour), d (day), w (week)。

sub_directory: 子目录名字。

base_filename: 文件前缀, 如 news, 则会依次将数据存储到文件 news_20110403_00000, news_20110403_00001 等。

2. buffer

这是最常用的一种 Store。该 Store 中包含两个子 Store, 其中一个 Primary Store, 另一个是 Secondary Store。日志会优先写到 Primary Store 中, 如果 Primary Store 出现故障, 则 Scribe 会将日志暂存到 Secondary Store 中, 待 Primary Store 恢复性能后, 再将 Secondary Store 中的数据复制到 Primary Store 中。其中, Secondary Store 仅支持两种 Store, 一个是 File, 另一个是 HDFS。

3. Null

这也是一种常用的 Store。用户可以在配置文件中配置一种叫 default 的 category, 如果数据所属的 category 没有在配置文件中设置相应的存储方式, 则该数据会被当作 default。如果用户想忽略这样的数据, 可以将它放入 Null store 中。

总起来说, Scribe 为日志收集提供了一种容错且可扩展的方案。Scribe 可以从不同数据源、不同机器上收集日志, 然后将它们存入一个中央存储系统, 以便于进一步处理。当采

用 HDFS 作为中央系统时，可以进一步利用 Hadoop 进行处理数据，于是 Scribe+HDFS+MapReduce 方案便诞生了，具体如图 5.5 所示。

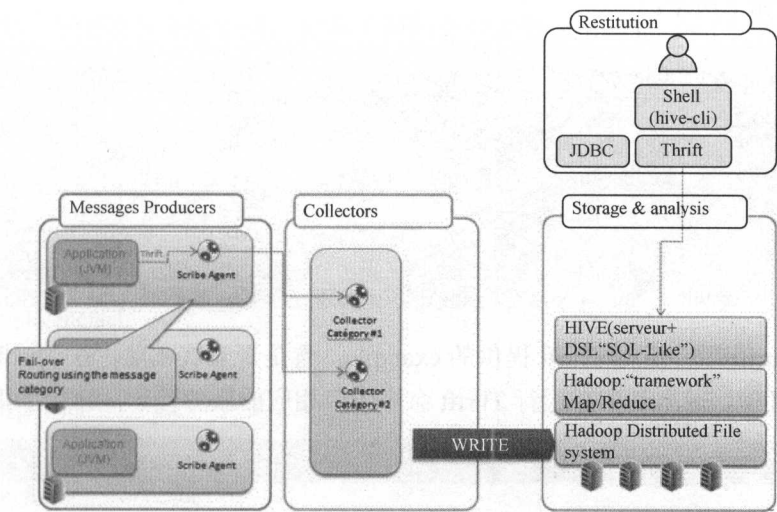


图 5.5 Scribe 收集方案

5.2.3 Scribe 环境搭建与部署

Scribe 的安装非常复杂，主要是因为其依赖的包需要设置的环境变量非常多，另外，它与 Hadoop 兼容不好，安装极需要技巧。本章根据实际安装经验，比较全面地介绍 Scribe 的安装方法。

1. 环境要求

- ① Ubuntu 12.04_64bit;
- ② Thrift 依赖软件;
- ③ Thrift;
- ④ Scribe;
- ⑤ Hadoop。

Apache 的所有版本（0.20.2 之前）不可用，因为它们 libhdfs 与 Scribe 均不兼容。Cloudera 的几乎所有版本不可用，只有 CDH3 beta2（0.20.2+320）可用。需要强调的是：Cloudera 的 0.20.2+737 和 0.20.2+320 的 HDFS 之间不能相互通信，Apache 0.20.2 版本和 Cloudera 0.20.2 之间也不能相互通信。

2. 安装流程

(1) 安装 Thrift 依赖软件

必须安装的是：g++、boost、autoconf、libevent、Apache ant、JDK、PHP 和 Python，

其他脚本语言根据需要安装。

(2) 安装 Thrift

主要步骤如下所示：

```
tar -zxvf thrift-0.2.0.tar.gz
cd thrift-0.2.0
./bootstrap.sh
./configure
sudo make
sudo make install
```

安装完 Thrift 后，运行 Thrift 提供的 example，看是否安装成功。在 Thrift 源代码目录找到 tutorial 目录，进入其中后运行 Thrift 命令生成相应的服务代码，命令如下：

```
$ thrift -r -gen cpp tutorial.thrift
```

运行完之后会在当前目录看到一个 gen-cpp 目录，其中就是 Thrift 命令生成的代码。进入 tutorial/cpp 目录，运行 make，生成相应的 CppServer 与 CppClient 程序。此时我们可以分别运行 CppServer 和 CppClient，让它们通信。

(3) 安装 Hadoop

如果发现 Hadoop 中自带的已经编译好的 libhdfs 不可用（libhdfs 在 \$HADOOP_HOME/c++ 中），可自己编译生成 libhdfs，方法是在 \$HADOOP_HOME 下，输入：ant compile-c++-libhdfs -Dislibhdfs=true，同时设置 Hadoop 的 CLASSPATH。

(4) 安装 Scribe

./bootstrap.sh（主要目的是生成 configure，如果出现类似于 configure: error: Could not link against! 的错误，可以暂且不管，直接进行下一步）

```
./configure -with-boost=/usr/local/boost -prefix=/usr/local/scribe
-with-hadooppath=/home/dong/hadoop-0.20.2/ -enable-hdfs CPPFLAGS="-I/opt/jdk1.6.0_21/include/
-I/opt/jdk1.6.0_21/include/linux -I/home/dong/hadoop-0.20.2/src/c++/libhdfs"
LDFLAGS="-L/opt/jdk1.6.0_21/jre/lib/amd64 -L/opt/jdk1.6.0_21/jre/lib/amd64/server
-L/home/dong/hadoop-0.20.2/build/c++/Linux-amd64-64/lib -ljvm -lhdfs"

make

sudo make install
```

3. 配置

至此, Scribe 和 Hadoop 相关软件安装已经完毕, 接下来是配置, 本章中, 我们介绍简单的配置方式, 方便读者快速完成, 按照下面的方式配置 Scribe 的.conf 文件。

```
#配置 Scribe 监听 1464 端口的消息, 并用文件的形式把日志放到./logs/目录下
port=1463
max_msg_per_second=2000000
check_interval=3
<store>
  category=default
  type=file
  fs_type=std
  file_path=./logs/
  base_filename=tmpdata
  max_size=1000000000
</store>
```

然后运行 `scribed-c scribe.conf` 启动 Scribe, 如果得到下列输出, 表示安装完成。

```
[Tue Nov 18 16:37:55 2014] "STATUS: STARTING"
[Tue Nov 18 16:37:55 2014] "STATUS: configuring"
.....
[Tue Nov 18 16:37:55 2014] "STATUS: ALIVE"
[Tue Nov 18 16:37:55 2014] "Starting scribe server on port 1463"
Thrift: Tue Nov 18 16:37:55 2014 TNonblockingServer: Serving on port 1463, 1 io threads.
Thrift: Tue Nov 18 16:37:55 2014 TNonblockingServer: using libevent 1.4.13-stable method epoll
Thrift: Tue Nov 18 16:37:55 2014 TNonblocking: IO thread #0 registered for listen.
Thrift: Tue Nov 18 16:37:55 2014 TNonblocking: IO thread #0 registered for notify.
Thrift: Tue Nov 18 16:37:55 2014 TNonblockingServer: IO thread #0 entering loop...
```

4. 可能出现的错误及解决方法

① 运行 `bootstrap.sh` 时, 产生以下错误:

```
checking whether the Boost::System library is available... yes
checking whether the Boost::Filesystem library is available... yes
configure: error: Could not link against !
```

当安装的 boost 目录不在默认的 `/usr` 目录下时, 用户需要配置 boost 安装目录, 如:

```
./configure --with-boost=/usr/local/boost --prefix=/usr/local/scribe
```

② 运行 examples 时，产生以下错误：

```
Traceback (most recent call last):
File "examples/scribe_cat", line 24, in <module>
from scribe import scribe
ImportError: No module named scribe
```

出现错误是因为 Python 找不到 scribe 模块，需要把 Scribe package 包含进来，如：

```
export PYTHONPATH="/usr/lib/python2.6/site-packages/"
```

③ Exception in thread "main" java.lang.NoClassDefFoundError: org/apache/ hadoop/conf/ Configuration

解决方法是将 Hadoop 的 classpath 加到环境变量中，如：

```
export CLASSPATH=$HADOOP_HOME/hadoop-core-0.20.2+320.jar
```

5.3 Chukwa

Chukwa 是一个开源的用于监控大型分布式系统的数据收集系统。这是构建在 Hadoop 的 HDFS 和 MapReduce 框架之上的，继承了 Hadoop 的可伸缩性和鲁棒性。Chukwa 还包含了一个强大和灵活的工具集，可用于展示、监控和分析已收集的数据。

5.3.1 Chukwa 的设计目标

如图 5.6 所示，Chukwa 的基本流程是 Agent 负责采集数据并传送给 Collector，Collector 收集 Agent 采集的数据并定时写入集群中，MapReduce Jobs 定时启动，负责把集群中的数据分类、排序、去重和合并。Chukwa 主要有以下设计目标：

- ① 架构清晰，能够快速部署；
- ② 收集的数据类型广泛，具有高扩展性；
- ③ 与 Hadoop 无缝集成，能完成海量数据的收集与整理。

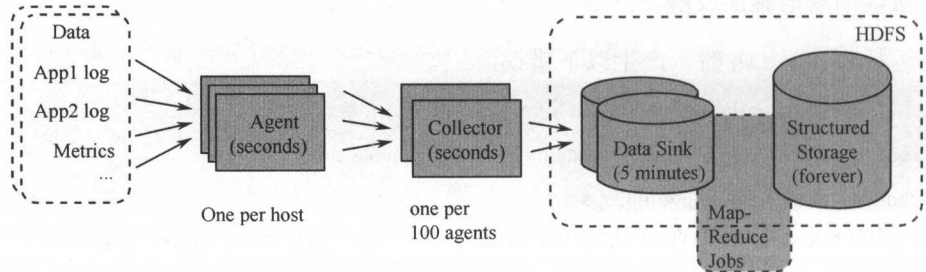


图 5.6 Chukwa 结构图

5.3.2 Chukwa 架构

Chukwa 是一个数据收集系统，它可以将各种各样类型的数据收集成适合 Hadoop 处理的文件保存在 HDFS 中供 Hadoop 进行各种 MapReduce 操作。Chukwa 本身也提供了很多内置的功能，帮助我们进行数据的收集和整理，架构如图 5.7 所示。

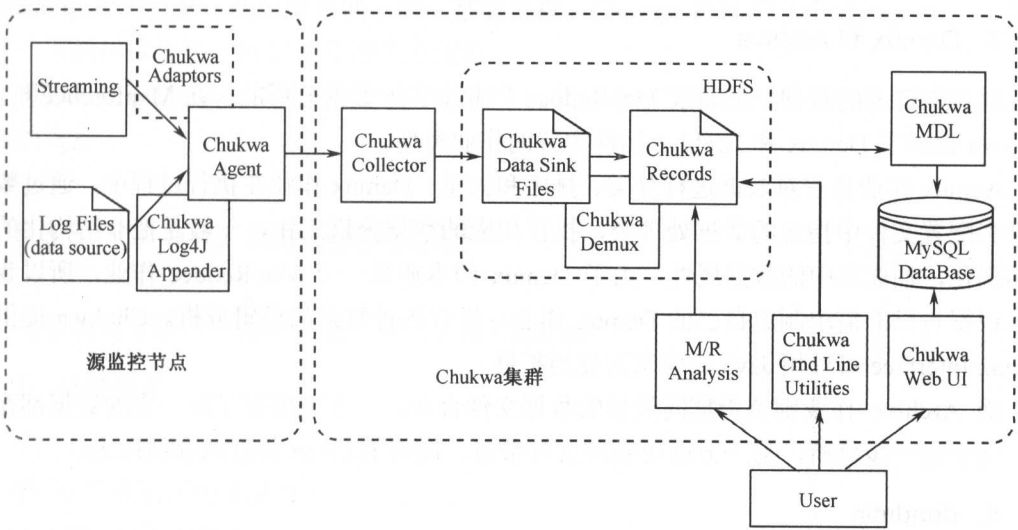


图 5.7 Chukwa 架构图

1. Adaptor 和 Agent

Agent 负责采集最原始的数据，并发送给 Collector，Adaptor 是直接采集数据的接口和工具，一个 Agent 可以管理多个 Adaptor 的数据采集，在每个数据的产生端（基本上是集群中每一个节点），Chukwa 使用一个 Agent 来采集它感兴趣的数据，每一类数据通过一个 Adaptor 来实现，数据的类型在相应的配置中指定。默认地，Chukwa 对以下常见的数据来源已经提供了相应的 Adaptor：命令行输出、log 文件和 httpSender 等。这些 Adaptor 会定期运行（比如每分钟读一次 df 的结果）或事件驱动地执行（比如 kernel 打了一条错误日志）。如果这些 Adaptor 还不够用，用户也可以方便地自己实现一个 Adaptor 来满足需求。

为防止数据采集端的 Agent 出现故障，Chukwa 的 Agent 采用了 watchdog 机制，会自动重启终止的数据采集进程，防止原始数据的丢失。另外，对于重复采集的数据，在 Chukwa 的数据处理过程中，会自动对它们进行去重。这样，就可以对于关键的数据在多台机器上部署相同的 Agent，从而实现容错的功能。

2. Collector

一方面，Collector 负责收集 Agent 送来的数据，并定时写入集群中，Agent 采集到的数据是存储到 Hadoop 集群上的。Hadoop 集群擅长于处理少量大文件，而对于大量小文件的

处理则不是它的强项，针对这一点，Chukwa 设计了 Collector 这个角色，用于把数据先进行部分合并，再写入集群，防止大量小文件的写入。

另一方面，为防止 Collector 成为性能瓶颈或成为单点，产生故障，Chukwa 允许和鼓励设置多个 Collector，Agent 随机地从 Collector 列表中选择一个 Collector 传输数据，如果一个 Collector 失败或繁忙，就换下一个 Collector。从而可以实现负载的均衡，实践证明，多个 Collector 的负载几乎是平均的。

3. Demux 和 Archive

放在集群上的数据，是通过 MapReduce 作业来实现数据分析的。在 MapReduce 阶段，Chukwa 提供了 Demux 和 Archive 两种内置的作业类型。

Aemux 作业负责对数据进行分类、排序和去重。Demux 作业在执行过程中，通过数据类型和配置文件中指定的数据处理类，执行相应的数据分析工作，一般是把非结构化的数据结构化，抽取其中的数据属性。由于 Demux 的本质是一个 MapReduce 作业，所以我们可以根据自己的需求制定自己的 Demux 作业，进行各种复杂的逻辑分析。Chukwa 提供的 Demux Interface 可以用 Java 语言来方便地扩展。

而 Archive 作业则负责把同类型的数据文件合并，一方面保证了同一类的数据都在一起，便于进一步分析，另一方面可减少文件数量，减轻 Hadoop 集群的存储压力。

4. dbadmin

放在集群上的数据，虽然可以满足数据的长期存储和大数据量计算需求，但是不便于展示。为此，Chukwa 做了两方面的努力。

① 使用 MDL 语言，把集群上的数据抽取到 MySQL 数据库中，对近一周的数据，完整保存，超过一周的数据，按数据离当前时间长短做稀释，离当前越久的数据，所保存的数据时间间隔越长。通过 MySQL 来做数据源，展示数据。

② 使用 HBase 或类似的技术，直接把索引化的数据存储于集群上。

5. Hicc

Hicc 是 Chukwa 的数据展示端的名字。在展示端，Chukwa 提供了一些默认的数据展示 Widget，可以使用列表、曲线图、多曲线图、柱状图、面积图等展示一类或多类数据，给用户展示直观的数据趋势。而且，在 Hicc 展示端，对不断生成的新数据和历史数据，采用 robin 策略，防止数据不断增长并增大服务器压力，对数据在时间轴上“稀释”，可以提供长时间段的数据展示。从本质上，Hicc 是用 Jetty 来实现的一个 Web 服务端，内部用的是 JSP 技术和 Javascript 技术。各种需要展示的数据类型和页面都可以通过简单的拖曳方式来实现，更复杂的数据展示方式，可以使用 SQL 语言组合出各种需要的数据。

6. 其他数据接口

如果对原始数据还有新的需要，用户还可以通过 MapReduce 作业或 Pig 语言直接访问

集群上的原始数据，以生成所需要的结果。Chukwa 还提供了命令行的接口，可以直接访问集群上的数据。

7. 默认数据支持

对于集群各节点的 CPU 使用率、内存使用率、硬盘使用率、集群整体的 CPU 平均使用率、集群整体的内存使用率、集群整体的存储使用率、集群文件数变化、作业数变化等等 Hadoop 相关数据，从采集到展示的一整套流程，Chukwa 都提供了内建的支持，只需要配置一下就可以使用，可以说是相当方便的。

可以看出，Chukwa 从数据的产生、收集、存储、分析到展示的整个生命周期都提供了全面的支持。

5.3.3 Chukwa 环境搭建与部署

这里我们介绍一下如何安装、部署应用 Chukwa。

1. 环境要求

- ① Linux 环境；
- ② 这里我们使用 Red Hat；
- ③ JDK 使用 JDK 1.6；
- ④ 系统中需要支持 SSH。

2. 下载 Chukwa

在本节中，我们使用 Chukwa 0.4.0 版本作为示例，可以用下面的命令下载其镜像：`wget http://mirror.bjtu.edu.cn/apache/hadoop/chukwa/chukwa-0.4.0/chukwa-0.4.0.tar.gz`。读者如果希望使用其他版本的 Chukwa，可以从官方网站下载（<http://incubator.apache.org/chukwa/>）。

3. 下载 Hadoop

Hadoop 的下载、安装不是本节重点，这里省略。在教程中，我们使用 0.20.2 版，能够很好地与 0.4.0 版本的 Chukwa 兼容，下载 Hadoop 时最重要的是考虑与当前版本 Chukwa 的兼容性。

4. 安装

- ① `tar -xzf chukwa-0.4.0.tar.gz`。
- ② `tar -xzf hadoop-0.20.2.tar.gz`。
- ③ 解压之后，假设目录名称分别为 Chukwa-0.4.0 和 Hadoop-0.20.2。

5. Hadoop 的配置

Hadoop 的配置参考官方配置实例即可，限于篇幅本节不再赘述。

6. Chukwa 的配置

本节中，我们将以最简单的步骤完成 Chukwa 的 Agent 和 Collector 的配置，使读者可以快速地了解它。

(1) 配置 Agent

编辑 \$CHUKWA_HOME/conf/chukwa-env.sh 文件，这里需要设置 JAVA_HOME 注释掉 HADOOP_HOME，HADOOP_CONF_DIR，因为 Agent 仅仅用来收集数据，所以不需要 HADOOP 的参与。注释掉 CHUKWA_PID_DIR，CHUKWA_LOG_DIR，如果不注释的话，那么它指定的位置在/tmp 临时目录下，这会导致 PID 和 LOG 文件被无故删除，会在后续的操作中导致异常。注释之后，系统会使用默认路径，默认会在 Chukwa 安装目录下创建 PID 和 LOG 文件。

编辑 \$CHUKWA_HOME/conf/collectors 文件，这里需要填写 Collector 的地址，格式为 http://hostname:port/，可以写多个 Collector，每个占一行。Agent 通常会随机选择一个作为 Collector 并将数据发送给这个 Collector，如果当前的 Collector 失败，则会继续选择下一个继续尝试。Collector 是有 loadbalance 功能的，不会出现所有的 Agent 都将数据写入一个 Collector，从而导致失败的情况。

编辑 \$CHUKWA_HOME/conf/initial_adapters 文件，将默认的配置文件 initial_adapters.template 改名为 initial_adapters。initial_adapters 文件是制定默认初始的 Adapter，当 Agent 启动的时候，这些 Adapter 就会工作。

(2) 配置 Collector

编辑 \$CHUKWA_HOME/conf/chukwa-env.sh 文件，修改 JAVA_HOME，HADOOP_HOME，HADOOP_CONF_DIR，指定为合适的值。同样的道理，我们需要注释掉 CHUKWA_PID_DIR，CHUKWA_LOG_DIR。

7. 启动

启动 Hadoop:

```
bin/start-all.sh
```

启动 Collector:

```
bin/chukwa collector
```

启动 Agent:

```
bin/chukwa agent
```

如果各项配置工作顺利完成，可以看到如下的日志结果：

清单 1. Agent 端日志片段

```

2014-11-13 10:20:28,315 INFO Timer-1 ExecAdaptor - calling exec
2014-11-13 10:20:28,377 INFO Timer-1 ExecAdaptor - calling exec
2014-11-13 10:20:28,438 INFO Timer-1 ExecAdaptor - calling exec
2014-11-13 10:20:28,451 INFO HTTP post thread ChukwaHttpSender -
                        collected 14 chunks for post_26923
2014-11-13 10:20:28,452 INFO HTTP post thread ChukwaHttpSender -
                        >>>>>> HTTP post_26923 to http://xi-pli:8080/ length = 17788
2014-11-13 10:20:28,459 INFO HTTP post thread ChukwaHttpSender -
                        >>>>>> HTTP Got success back from http://xi-pli:8080/chukwa; response length 924
2014-11-13 10:20:28,459 INFO HTTP post thread ChukwaHttpSender -
                        post_26923 sent 0 chunks, got back 14 acks
2014-11-13 10:20:28,500 INFO Timer-1 ExecAdaptor - calling exec

```

从这里我们能看到 Timer-1 ExecAdaptor - calling exec 已经被定时执行，而且我们还能看到 Agent 将信息发送给了我们指定的 Collector。

清单 2. Collector 端日志

```

2014-11-13 10:30:22,207 INFO Timer-4 SeqFileWriter -
        rotating sink file
/chukwa/logs/201423102522181_xipli_15db999712d1106ead87ffe.chukwa
2014-11-13 10:30:22,784 INFO Timer-1 root -
        stats:ServletCollector,numberHTTPConnection:15,numberchunks:1110
2014-11-13 10:30:23,220 INFO Timer-3 SeqFileWriter -
        stat:datacollection.writer.hdfs dataSize=797670 dataRate=26587

```

从日志我们可以看到 Collector 已经通过 writer 将收集到的数据写入了文件 /chukwa/logs/201423102522181_xipli_15db999712d1106ead87ffe.chukwa。

那么怎么知道数据已经被写入 HDFS 了呢？我们可以通过执行 Hadoop 的命令来查看 HDFS。

清单 3. 查看日志

```
bin/hadoop fs -ls /chukwa/logs
```

如果上述操作顺利，我们能看到数据已经被写入了 HDFS。

清单 4. 检查文件

```

Found 205 items
-rw-r--r--    3 hadoop supergroup
676395 2014-11-13 17:12
/chukwa/logs/201422171225851_xipli_18flec1212d0d4c13067ffa.done
-rw-r--r--    3 hadoop supergroup 6046366 2010-12-22 17:17
/chukwa/logs/201422171725877_xipli_18flec1212d0d4c13067ff8.chukwa
-rw-r--r--    3 hadoop supergroup
8352420 2014-11-13 17:32
/chukwa/logs/201422173249756_xipli_1f33c45712d0d6c7cdd8000.done

```

如果想要将 Agent 部署到多个节点去，只需要在其他节点上也做相应的配置即可。但是随着节点数目的增多，我们会发现管理 Agent 的难度越来越高。这时我们可以使用单一的节点来批量管理所有的 Agent。

我们首先需要编辑的文件是 conf 目录下的 Agent 文件，默认情况下，它是一个模板文件，名称为 agents.template，我们需要把它改名为 agents。将 Agent 的 hostname/IP 逐个添加进这个文件中，每行一个节点。这时通过调用 bin/start-agents.sh 和 bin/stop-agents.sh 可以批量地管理 Agent 的启动和关闭。如果遇到不能正常关闭 Agent 的情况，那么一个可用的临时解决方法是将各个节点的 Chukwa 脚本中的 kill -1 修改为 kill -9。Collector 也可用类似的控制方式。

5.4 Kafka

Kafka 是 Linkedin 用于日志处理的分布式消息队列，Linkedin 的日志数据容量大，但对可靠性要求不高，其日志数据主要包括用户行为（登录、浏览、点击、分享、喜欢）以及系统运行日志（CPU、内存、磁盘、网络、系统及进程状态）。

当前很多的消息队列服务提供可靠交付保证，并默认是即时消费（不适合离线）。高可靠交付对 Linkedin 的日志不是必需的，故可通过降低可靠性来提高性能，同时通过构建分布式的集群，允许消息在系统中累积，使得 Kafka 同时支持离线和在线日志处理。

5.4.1 Kafka 架构

Kafka 实际上是一个消息发布订阅系统。Producer 向某个 Topic 发布消息，而 Consumer 订阅某个 Topic 的消息，进而一旦有新的关于某个 Topic 的消息，Broker 会传递给订阅它的所有 Consumer。在 Kafka 中，消息是按 Topic 组织的，而每个 Topic 又会分为多个 Partition，

这样便于管理数据和进行负载均衡。同时，它也使用了 ZooKeeper 进行负载均衡。

Kafka 中主要有三种角色，分别为 Producer、Broker 和 Consumer，如图 5.8 所示。

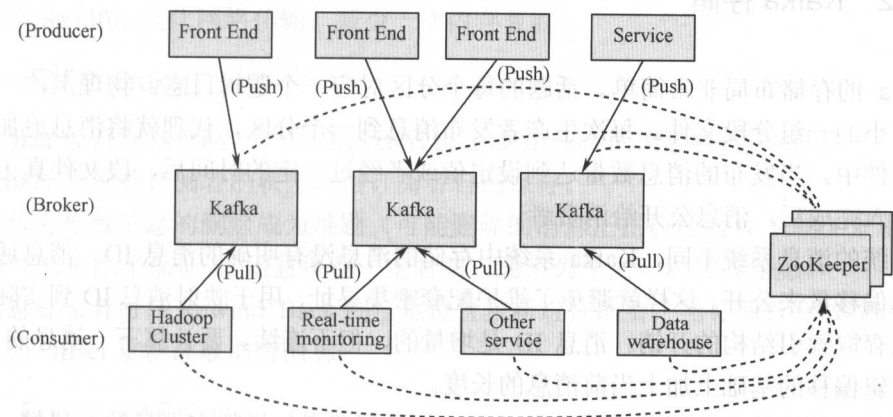


图 5.8 Kafka 架构图

1. Producer

Producer 的任务是向 Broker 发送数据。Kafka 提供了两种 Producer 接口，一种是 Low level 接口，使用该接口会向特定的 Broker 的某个 Topic 下的某个 Partition 发送数据；另一种是 High level 接口，该接口支持同步/异步发送数据，基于 ZooKeeper 的 Broker 自动识别和进行负载均衡（基于 Partitioner）。

Producer 可以通过 ZooKeeper 获取可用的 Broker 列表，也可以在 ZooKeeper 中注册 Listener，该 Listener 在以下情况下会被唤醒。

- ① 添加一个 Broker;
- ② 删除一个 Broker;
- ③ 注册新的 Topic;
- ④ Broker 注册已存在的 Topic。

当 Producer 得知以上事件时，可根据需要采取一定的行动。

2. Broker

Broker 采取了多种策略提高数据处理效率，包括 Sendfile 和 zero copy 等技术。

3. Consumer

Consumer 的作用是将日志信息加载到中央存储系统上。Kafka 提供了两种 Consumer 接口，一种是 Low level 的，它维护到某一个 Broker 的连接，并且这个连接是无状态的，即每次从 Broker 上 pull 数据时，都要告诉 Broker 数据的偏移量。另一种是 High level 接口，它隐藏了 Broker 的细节，允许 Consumer 从 Broker 上 push 数据而不必关心网络拓扑结构。更重要的是，对于大部分日志系统而言，Consumer 已经获取的数据信息都由 Broker 保存，

而在 Kafka 中，由 Consumer 自己维护所取数据信息。

5.4.2 Kafka 存储

Kafka 的存储布局非常简单。话题的每个分区对应一个逻辑日志。物理上，一个日志为相同大小的一组分段文件。每次生产者发布消息到一个分区，代理就将消息追加到最后一段文件中。当发布的消息数量达到设定值或者经过一定的时间后，段文件真正写入磁盘中。写入完成后，消息公开给消费者。

与传统的消息系统不同，Kafka 系统中存储的消息没有明确的消息 ID。消息通过日志中的逻辑偏移量来公开。这样就避免了维护配套密集寻址，用于映射消息 ID 到实际消息地址的随机存取索引结构的开销。消息 ID 是增量的，但不连续。要计算下一消息的 ID，可以在其逻辑偏移的基础上加上当前消息的长度。

消费者始终从特定分区顺序地获取消息，如果消费者知道特定消息的偏移量，也就说明消费者已经消费了之前的所有消息。消费者向代理发出异步请求，准备字节缓冲区用于消费。每个异步请求都包含要消费的消息偏移量。Kafka 利用 Sendfile API 高效地从代理的日志文件中分发字节给消费者（图 5.9）。

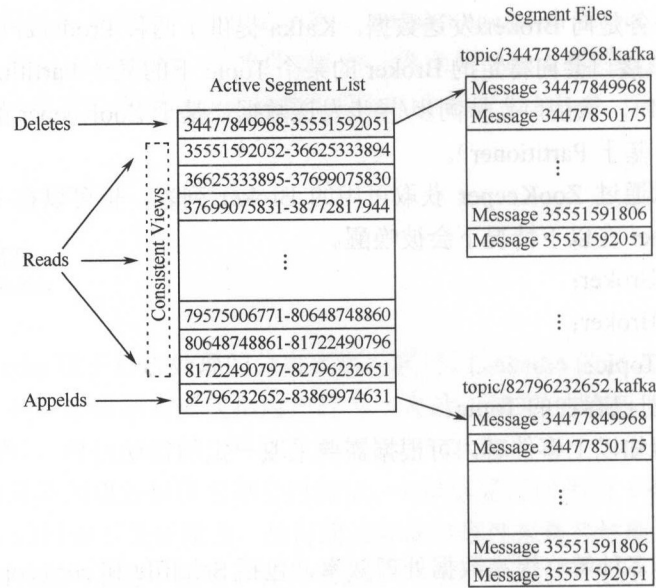


图 5.9 Kafka 存储架构

5.4.3 Kafka 的特点

1. 高效的数据传输

发布者每次可发布多条消息(将消息加到一个消息集合中发布), Sub 每次迭代一条消息。

不创建单独的 Cache，使用系统的 Page Cache。发布者顺序发布，订阅者通常比发布者滞后一点点，直接使用 Linux 的 page Cache 效果也比较好，同时减少了 Cache 管理及垃圾收集的开销。

使用 Sendfile 优化网络传输，减少一次内存复制。

2. 无状态 Broker

Broker 没有副本机制，一旦 Broker 宕机，该 Broker 的消息将都不可用。

Broker 不保存订阅者的状态，由订阅者自己保存。

无状态导致消息的删除成为难题（可能删除的消息正在被订阅），Kafka 采用基于时间的 SLA（服务水平保证），消息保存一定时间（通常为 7 天）后会被删除。

消息订阅者可以 Rewind Back 到任意位置重新进行消费，当订阅者有故障时，可以选择最小的 Offset 重新读取消费消息。

3. Consumer Group

允许 Consumer Group（包含多个 Consumer，如一个集群同时消费）对一个 Topic 进行消费，不同的 Consumer Group 之间独立订阅。

为了减小一个 Consumer Group 中不同 Consumer 之间的分布式协调开销，指定 Partition 为最小的并行消费单位，即一个 Group 内的 Consumer 只能消费不同的 Partition。

4. ZooKeeper 协调控制

管理 Broker 与 Consumer 的动态加入与离开。

触发负载均衡，当 Broker 或 Consumer 加入或离开时会触发负载均衡算法，使得一个 Consumer Group 内的多个 Consumer 的订阅负载均衡。

维护消费关系及每个 Partiton 的消费信息。

5. ZooKeeper 上的细节

每个 Broker 启动后会在 ZooKeeper 上注册一个临时的 Broker Registry，包含 Broker 的 IP 地址和端口号，所存储的 Topics 和 Partition 信息。

每个 Consumer 启动后会在 ZooKeeper 上注册一个临时的 Consumer Registry，包含 Consumer 所属的 Consumer Group 以及订阅的 Topic。

每个 Consumer Group 关联一个临时的 Owner Registry 和一个持久的 Offset Registry。对于被订阅的每个 Partition 包含一个 Owner Registry，内容为订阅这个 Partition 的 Consumer ID；同时包含一个 Offset Registry，内容为上一次订阅的 Offset。

6. 消息交付保证

Kafka 对消息的重复、丢失、错误以及顺序没有严格的要求。

Kafka 提供 At-least-once Delivery，即当 Consumer 宕机后，有些消息可能会被重复 Delivery。

因为每个 Partition 只会被 Consumer Group 内的一个 Consumer 消费，故 Kafka 保证每

个 Partition 内的消息会被顺序地订阅。

Kafka 为每条消息计算 CRC 校验，用于错误检测，CRC 校验不通过的消息会直接被丢弃。

5.4.4 Kafka 环境搭建与部署

1. 环境要求

- ① Ubuntu 12.04_64bit;
- ② Kafka 依赖文件;
- ③ Kafka;
- ④ ZooKeeper。

2. 安装流程

下载后的 Kafka 包进行解压并编译，由于 Kafka 核心编程语言为 Scala，这里我们需要使用 sbt 命令，因为需要下载很多依赖文件，./sbt update 命令会花费较多时间。

```
# tar xzvf kafka-0.8.0-beta1-src.tgz
# cd kafka-0.8.0-beta1-src
# ./sbt update
# ./sbt package
# ./sbt assembly-package-dependency
```

3. 启动服务

首先启动 ZooKeeper 服务，可以使用 Kafka 提供的单节点脚本。

```
# bin/ZooKeeper-server-start.sh config/ZooKeeper.properties &
```

&是为了在后台运行，否则还要手动放于后台或者重新开启一个终端窗口。然后启动 Kafka 服务，同样使用 Kafka 自身提供的脚本。

```
# bin/kafka-server-start.sh config/server.properties &
```

4. 创建 Topic

可以使用下面的命令创建一个 Topic，名字叫 test。

```
# bin/kafka-create-topic.sh --ZooKeeper localhost:2181 --replica 1 --partition 1 --topic test
```

使用下面的命令可以查看创建的 Topic，使用 2>/dev/null 屏蔽一些线程的启动信息等，只显示关键结果。

```
# bin/kafka-list-topic.sh --ZooKeeper localhost:2181 2>/dev/null
topic: test    partition: 0    leader: 0    replicas: 0    isr: 0
```


5. 收发消息

Kafka 发送的消息可以是文件或者标准输入的一行，使用如下命令发送 Kafka 消息。

```
#bin/kafka-console-producer.sh --broker-list localhost:9092 --topic test >/dev/null
This is a test message
```

可以使用如下命令接受 Kafka 消息。

```
# bin/kafka-console-consumer.sh --ZooKeeper localhost:2181 --topic test --from-beginning 2>/dev/null
This is a test message
```

5.5 练习题

1. 大数据应用中数据采集系统需要具备的特点有哪些？
2. Flume 系统中 Agent、Collector、Master 的功能分别是什么？
3. Scribe 系统的独特优势是什么？
4. Kafka 的特点有哪些？

参考文献

- [1] <http://flume.apache.org/>
- [2] <http://www.cnblogs.com/oubo/archive/2012/05/25/2517751.html>
- [3] <https://github.com/facebookarchive/scribe/wiki>
- [4] Thusoo, Ashish, et al. "Data warehousing and analytics infrastructure at facebook." Proceedings of the 2010 ACM SIGMOD International Conference on Management of data. ACM, 2010.
- [5] <https://chukwa.apache.org/>
- [6] Rabkin, Ariel, and Randy H. Katz. "Chukwa: A System for Reliable Large-Scale Log Collection." LISA. Vol. 10. 2010.
- [7] <http://www.ibm.com/developerworks/cn/opensource/os-cn-chukwa/>
- [8] <http://sna-projects.com/kafka/>
- [9] Kreps, Jay, Neha Narkhede, and Jun Rao. "Kafka: A distributed messaging system for log processing." Proceedings of the NetDB. 2011.
- [10] <http://dongxicheng.org/search-engine/log-systems/>
- [11] <http://dongxicheng.org/search-engine/kafka/>

第6章

Hadoop 与 MapReduce

本章将简要介绍 Hadoop 分布式生态系统的各组成部分，首先从 Hadoop 的底层文件系统 HDFS 开始，接着介绍了 Hadoop 的结构化存储系统 HBase，重点讲解 Hadoop 的任务调度框架 MapReduce 的原理，并给出了一个具体的编程实例，对 MapReduce 2.0 做简单的介绍和对比。由于篇幅有限，在 6.3 节简要介绍 Hadoop 生态圈的几个重要组成部分的功能。最后介绍 Hadoop 平台的应用案例。

6.1 Hadoop 平台

6.1.1 Hadoop 概述

Hadoop 是 Apache 软件基金会旗下的一个开源分布式计算平台。它的核心部分由 Hadoop 分布式文件系统 HDFS (Hadoop Distributed File System) 和 MapReduce (Google MapReduce 的开源实现) 组成，为用户提供了系统底层细节透明的分布式基础架构。其中 HDFS 的高容错性、高伸缩性、高可用性等优点允许用户将 Hadoop 部署在普通的个人电脑上，形成分布式系统。MapReduce 分布式编程模型允许用户在不了解分布式系统底层细节的情况下开发并行应用程序。所以，用户可以利用 Hadoop 轻松地组织计算机资源，搭建自己的分布式计算平台，并且可以充分利用集群的计算和存储能力，完成海量数据的处理。

Hadoop 被公认是一套行业大数据标准开源软件，在分布式环境下提供了海量数据的处理能力。国内外很多大公司都在利用 Hadoop 处理公司业务，也有很多公司围绕 Hadoop 做

工具开发、开源软件和技术服务。一方面,2013年,大型IT公司,如EMC、Microsoft、Intel、Teradata、Cisco都明显增加了Hadoop方面的投入,Teradata还公开展示了一个一体机;另一方面,创业型Hadoop公司层出不穷,如Sqrrl、Wandisco、GridGain、InMobi等,都推出了开源的或者商用的软件。

目前有很多公司开始提供基于Hadoop的商业软件、支持、服务以及培训。Cloudera是一家美国的企业软件公司,该公司在2008年开始提供基于Hadoop的软件和服务。Hortonworks是由Yahoo和Benchmark Capital于2011年7月联合创建的一家企业管理软件公司,专注于Apache Hadoop的开发和支持,支持跨计算机集群分布式处理大型数据集,主要产品为Hortonworks数据平台(一款开源的基于Apache Hadoop的数据分析系统),该公司雇佣了众多Hadoop项目的核心人员以提供相应的支持和培训。GoGrid是一家云计算基础设施公司,在2012年,该公司与Cloudera合作加速了企业采纳基于Hadoop应用的步伐。Dataguise公司是一家数据安全公司,2012年该公司推出了一款针对Hadoop的数据保护和风险评估软件。

6.1.2 Hadoop的发展简史

Hadoop是Doug Cutting开发的使用广泛的文本搜索库。Hadoop起源于Apache Nutch——一个开源的网络搜索引擎,本身也是Lucene项目的一部分。Hadoop这个名字不是一个缩写,它是一个虚构的名字。该项目的创建者Doug Cutting如此解释Hadoop的得名:“这个名字是我孩子给一头吃饱了的棕黄色大象起的名字。我的命名标准就是简短,容易发音和拼写,没有太多的意义,并且不会被用于别处。小孩子是这方面的高手。Google就是由小孩命名的。”Hadoop及其子项目和后继模块所使用的名字往往也与其功能不相关,经常用一头大象和其他动物主题(例如Pig、Hive)。然而各个较小的组成部分的命名体现了更多的描述性,更通俗易懂,因为它意味着可以大致从其名字猜测其功能,例如,JobTracker的任务就是跟踪MapReduce作业。

从头开始构建一个网络搜索引擎是一个雄心勃勃的目标,不只是一要编写一个复杂的、能够抓取和索引网站的软件,还需要面临着没有专业运行团队支持运行它的挑战,因为它有那么多独立部件。同样昂贵的还有硬件设施:据Mike Cafarella和Doug Cutting估计,一个支持10亿页的索引需要价值约为50万美元的硬件投入,每月运行费用还需要3万美元。不过,他们相信这是一个有价值的目标,因为这会开放并最终使搜索引擎算法普及化。

Nutch项目开始于2002年,一个可工作的抓取工具和搜索系统很快浮出水面。但他们意识到,他们的架构将无法扩展到拥有数十亿网页的网络。在2003年发表的一篇描述Google分布式文件系统(Google File System, GFS)的论文为他们提供了及时的帮助,文中称Google正在使用此文件系统。GFS或类似的文件系统,可以解决他们在网络抓取和索引过程中产生的大量的文件存储需求。具体而言,GFS会省掉管理所花的时间,如管理存储节点。在2004年,他们开始写一个开放源码的应用,即Nutch的分布式文件系统(NDFS)。

2004 年, Google 发表了论文, 向全世界介绍了 MapReduce。2005 年年初, Nutch 的开发者在 Nutch 上有了一个可工作的 MapReduce 应用, 到当年年中, 所有主要的 Nutch 算法被移植到 MapReduce 和 NDFS 上运行。

Nutch 中的 NDFS 和 MapReduce 实现的应用远不只在搜索领域, 2006 年 2 月, 他们从 Nutch 转移出来成为一个独立的 Lucene 子项目, 称为 Hadoop。大约在同一时间, Doug Cutting 加入雅虎, Yahoo 提供一个专门的团队和资源将 Hadoop 发展成一个可在网络上运行的系统。2008 年 2 月, 雅虎宣布其搜索引擎产品部署在一个拥有 1 万个内核的 Hadoop 集群上。

2008 年 1 月, Hadoop 已成为 Apache 顶级项目, 证明它是成功的, 是一个多样化、活跃的社区。通过这次机会, Hadoop 成功地被雅虎之外的很多公司应用, 如 Last.fm、Facebook 和《纽约时报》。

《纽约时报》使用亚马逊的 EC2 云计算将 4TB 的报纸扫描文档压缩, 转换为用于 Web 的 PDF 文件。这个过程历时不到 24 小时, 使用 100 台机器运行, 如果不结合亚马逊的按小时付费的模式(即允许《纽约时报》在很短的一段时间内访问大量机器)和 Hadoop 易于使用的并程序序设计模型, 该项目很可能不会这么快开始启动。

2008 年 4 月, Hadoop 打破世界纪录, 成为最快排序 1TB 数据的系统。运行在一个 910 节点的群集, Hadoop 在 209 秒内排序了 1TB 的数据(还不到三分半钟), 击败了前一年的 297 秒的冠军。同年 11 月, 谷歌在报告中声称, 它的 MapReduce 实现 1TB 数据的排序只用了 68 秒。2009 年 5 月, 有报道宣称 Yahoo 的团队使用 Hadoop 对 1TB 的数据进行排序只花了 62 秒。

6.1.3 Hadoop 的功能和作用

21 世纪, 我们已经迈入了大数据时代, 众所周知, 现代社会的信息量增长速度极快, 这些信息里又积累着大量的数据, 其中包括个人数据和工业数据。根据 Facebook 在 2012 年公开的一组数据显示, 诸如 Facebook 系统每天要处理 25 亿条消息、500 TB 以上的数据、用户点击 Like 按钮的次数达到 27 亿次、上传 3 亿张照片、每半个小时扫描的数据大约为 105TB。预计到 2020 年, 每年产生的数字信息将会有超过 1/3 的内容驻留在云平台中或借助云平台处理。我们需要对这些数据进行分析 and 处理, 以获取更多有价值的信息。那么我们如何高效地存储和管理这些数据, 如何分析这些数据呢? Hadoop 作为开源分布式大数据处理平台的价值在这个时候就体现出来了, 它在处理这类问题时, 采用了分布式存储方式, 提高了读写速度, 并扩大了存储容量。采用 MapReduce 来整合分布式文件系统上的数据, 可保证分析和处理数据的高效。与此同时, Hadoop 还采用存储冗余数据的方式保证了数据的安全性。

Hadoop 中 HDFS 的高容错特性, 以及它是基于 Java 语言开发的, 这使得 Hadoop 可以部署在低廉的计算机集群中, 同时不限于某个操作系统。Hadoop 中 HDFS 的数据管理能力, MapReduce 处理任务时的高效率, 以及它的开源特性, 使其在同类的分布式系统中大放异

彩，并在众多行业和科研领域中被广泛采用。

Hadoop 是一个能够让用户轻松架构和使用的分布式计算平台。用户可以轻松地在 Hadoop 上开发和运行处理海量数据的应用程序。它主要有以下几个优点。

- ① 高可靠性: Hadoop 按位存储和处理数据的能力值得人们信赖。
- ② 高扩展性: Hadoop 是在可用的计算机集簇间分配数据并完成计算任务的, 这些集簇可以方便地扩展到数以千计的节点中。
- ③ 高效性: 与生俱来的并行化处理方式使得 Hadoop 能够在节点之间动态地移动数据, 并保证各个节点的动态平衡, 还能方便地扩展集群, 增大机器规模, 因此其处理速度非常快。
- ④ 高容错性: Hadoop 能够自动保存数据的多个副本, 并且能够自动将失败的任务重新分配。
- ⑤ 低成本: 组成 Hadoop 平台的机器都是廉价的服务器甚至是个人电脑, 成本非常低。

6.1.4 HDFS

Hadoop 由许多元素构成, 其底部是 Hadoop Distributed File System (HDFS), 它存储 Hadoop 集群中所有存储节点上的文件, 是 GFS 的开源实现, 因此本节只对 HDFS 做简要的介绍, 包括 HDFS 的特点、架构、读写操作过程, 具体的原理可以参考第 3 章对 GFS 的详细介绍。

HDFS 是一种分布式文件系统, 运行于大型商用机集群, HDFS 为 HBase 提供了高可靠性的底层存储支持; 由于 HDFS 具有高容错性的特点, 所以可以设计部署在低廉的硬件上。它可以以很高的吞吐率来访问应用程序的数据, 适合那些有着超大数据集的应用程序。HDFS 与其他分布式文件系统有许多相似点, 但也有几个不同点。一个明显的区别是 HDFS 的“一次写入多次读取”(write-once-read-many)模型, 该模型降低了并发性控制要求, 简化了数据聚合性, 支持高吞吐量访问。

HDFS 的另一个独特的特性是下面这个观点: 将处理逻辑放置到数据附近通常比将数据移向应用程序空间更好(移动程序比移动数据更划算)。通常一个数据处理程序只有几 KB 至几 MB 的大小, 而数据则非常大, 显然, 将程序移动到数据所在的位置, 处理完数据之后将处理结果传回调用方, 这样能节省很多网络带宽资源。

HDFS 将数据写入严格限制为一次写入一个。字节总是被附加到一个流的末尾, 字节流总是以写入顺序存储。

HDFS 有许多目标, 下面是一些最明显的目标。

① 通过检测故障和应用快速、自动地恢复实现容错性: 由于 HDFS 建立在大量普通的硬件设备上, 因此硬件故障是常见的问题, 整个 HDFS 系统由数百台或数千台存储着数据文件的服务器组成, 而如此多的服务器意味着高故障率, 所以故障的检测和自动快速恢复是 HDFS 的一个核心目标。

- ② 通过 MapReduce 流进行数据访问：HDFS 使应用程序能流式地访问它们的数据集。HDFS 被设计成适合进行批量处理，而不是用户交互式的处理。所以它重视数据吞吐量，而不是数据访问的反应速度。
- ③ 简单可靠的聚合模型。
- ④ 处理逻辑接近数据，而不是数据接近处理逻辑。
- ⑤ 跨异构普通硬件和操作系统的可移植性。
- ⑥ 可靠存储和处理大量数据的可伸缩性。
- ⑦ 通过跨多个普通个人计算机集群分布数据和处理来节约成本。
- ⑧ 通过分布数据和逻辑到数据所在的多个节点上进行平行处理来提高效率。
- ⑨ 通过自动维护多个数据副本和在故障发生时自动重新部署处理逻辑来实现可靠性。

HDFS 是分布式计算的存储基础，Hadoop 的分布式文件系统和其他分布式文件系统有很多类似的特质（图 6.1）。分布式文件系统的几个基本特点如下。

- ① 对于整个集群有单一的命名空间。
- ② 数据一致性。适合一次写入多次读取的模型，客户端在文件没有被成功创建之前无法看到文件存在。
- ③ 文件会被分割成多个文件块，每个文件块被分配存储到数据节点上，而且根据配置会由复制文件块来保证数据的安全性。

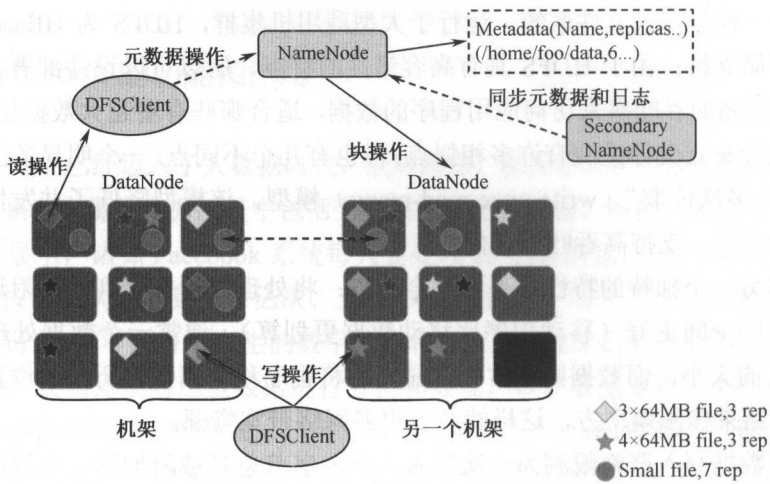


图 6.1 HDFS 系统架构图

图 6.1 展现了 HDFS 的三个重要角色：NameNode、DataNode 和 Client。NameNode 可以看成分布式文件系统管理者，主要负责管理文件系统的命名空间、集群配置信息和存储块的复制等。NameNode 会将文件系统的 Metadata 存储在内存中，这些信息主要包括

了文件信息、每一个文件对应的文件块的信息和每一个文件块在 **DataNode** 中的信息等。**DataNode** 是文件存储的基本单元，它将 **Block** 存储在本地文件系统中，保存了 **Block** 的 **Metadata**，同时周期性地将所有存在的 **Block** 信息发送给 **NameNode**。**Client** 就是需要获取分布式文件系统中的文件的应用程序。这里通过三个操作来说明它们之间的交互关系。

1. 文件写入操作（图 6.2）

文件写入的步骤：

- ① **Client** 向 **NameNode** 发起文件写入的请求。
- ② **NameNode** 根据文件大小和文件块配置情况，返回给 **Client** 它所管理部分 **DataNode** 的信息。
- ③ **Client** 将文件划分为多个 **Block**，根据 **DataNode** 的地址信息，按顺序写入每一个 **DataNode** 块中。

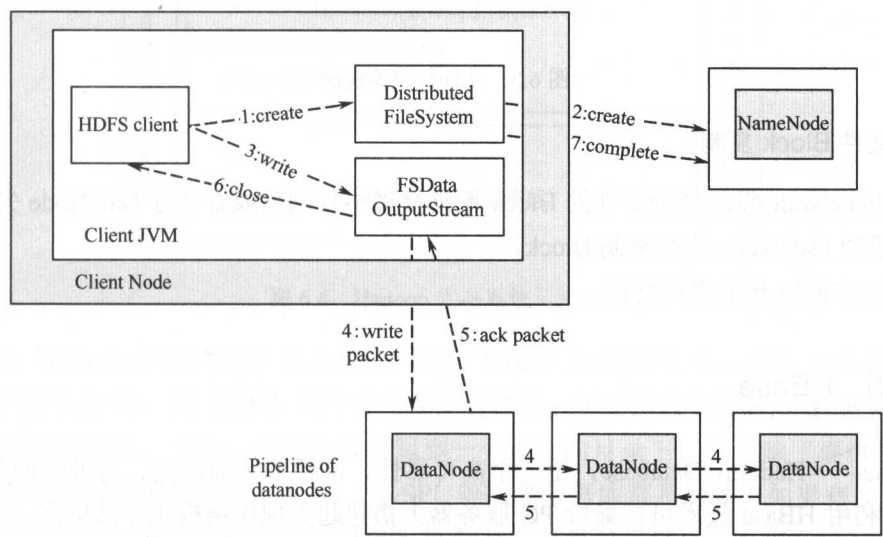


图 6.2 文件写入流程图

2. 文件读取操作（图 6.3）

文件读取步骤：

- ① **Client** 向 **NameNode** 发起文件读取的请求。
- ② **NameNode** 返回文件存储的 **DataNode** 的信息。
- ③ **Client** 读取文件信息。

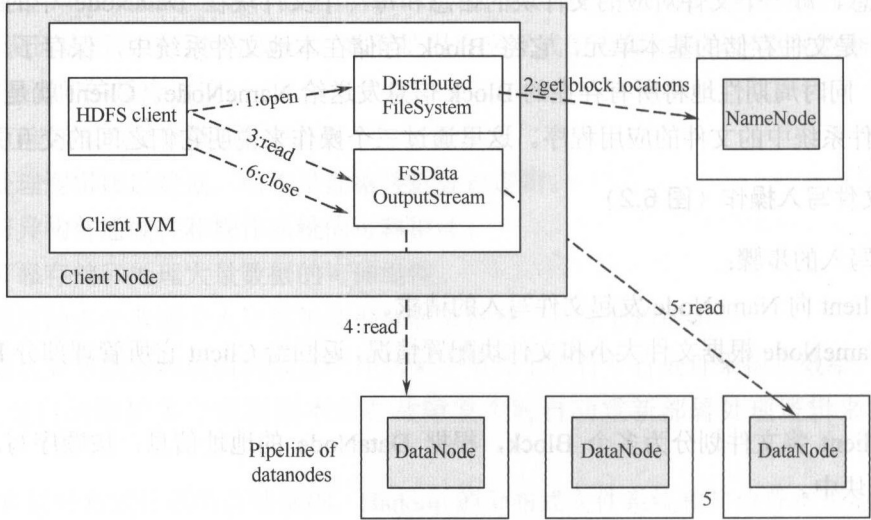


图 6.3 文件读取流程图

3. 文件 Block 复制

- ① NameNode 发现部分文件的 Block 不符合最小复制数或者部分 DataNode 失效。
- ② 通知 DataNode 相互复制 Block。
- ③ DataNode 开始相互复制。

6.1.5 HBase

HBase——Hadoop Database，是一个高可靠性、高性能、面向列、可伸缩的分布式存储系统，利用 HBase 技术可在廉价 PC 服务器上搭建起大规模结构化存储集群。HBase 是 Google BigTable 的开源实现，模仿并提供了基于 Google 文件系统的 BigTable 数据库的所有功能。Google BigTable 利用 GFS 作为其文件存储系统，HBase 利用 Hadoop HDFS 作为其文件存储系统；Google 运行 MapReduce 来处理 BigTable 中的海量数据，HBase 同样利用 Hadoop MapReduce 来处理 HBase 中的海量数据；Google BigTable 利用 Chubby 作为协同服务，HBase 利用 ZooKeeper 作为协同服务。HBase 仅能通过行键（row key）和行键的值域区间范围（range）来检索数据，并且仅支持单行事务（可通过 Hive 支持来实现多表连接等复杂操作）。HBase 主要用来存储非结构化和半结构化的松散数据。HBase 可以直接使用本地文件系统或者 Hadoop 作为数据存储方式，不过为了提高数据可靠性和系统的健壮性，发挥 HBase 处理大数据量等功能，需要使用 Hadoop 作为文件系统。与 Hadoop 一样，HBase 主要依靠横向扩展，通过不断增加廉价的商用服务器来增加计算和存储能力。HBase 的目标是处理非常庞大的表，可以用普通的计算机处理超过 10 亿行数据并且由数百万列元素组成的数据表。HBase 中的表一般有这样的特点：

- ① 大：一个表可以有上亿行，上百万列。
- ② 面向列：面向列（族）的存储和权限控制，列（族）独立检索。
- ③ 稀疏：对于为空（null）的列，并不占用存储空间，因此，表可以设计得非常稀疏。

图 6.4 描述了 Hadoop 生态系统中的各层系统，其中，HBase 位于结构化存储层，HDFS 为 HBase 提供了高可靠性的底层存储支持，MapReduce 为 HBase 提供了高性能的计算能力，ZooKeeper 为 HBase 提供了稳定服务和失败恢复机制。此外，Pig 和 Hive 还为 HBase 提供了高层语言支持，使得在 HBase 上进行数据统计处理变得非常简单。Sqoop 则为 HBase 提供了方便的 RDBMS 数据导入功能，使得传统数据库数据向 HBase 中迁移变得非常方便。

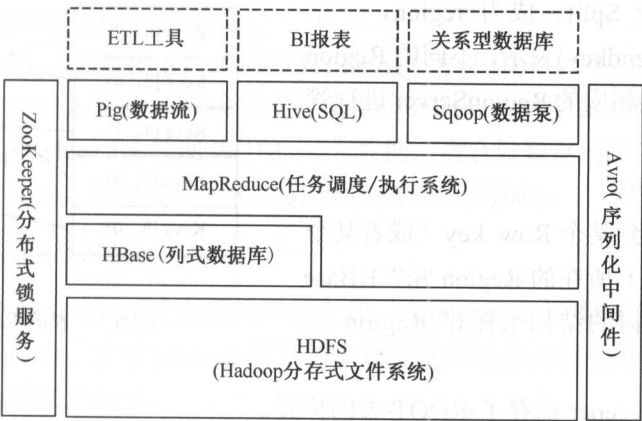


图 6.4 Hadoop 生态系统示意图

HBase 与 Hadoop 之间是什么关系呢？其实 HBase 跟 HDFS 没有必然的关系，HBase 是对数据关系的管理，而 HDFS 是对文件存储的管理，可以想象如果没有 HDFS，HBase 会把用户请求的增加、更新、删除的数据写到一个固定的目录下，自己除了要维护数据关系外还要维护物理数据的存储和备份，MySPL 等很多数据库都是这样。如果有了 HDFS，HBase 就可以把要存储的数据发送到 HDFS 集群上，由 HDFS 进行分布式存储；如果说有关系，那就是 HBase 利用了 HDFS，HBase 基于 HDFS 之上。

1. HBase 数据模型

(1) 表和列簇（表 6.1）

表 6.1 表和列簇

Row Key	Timestamp	Column Family	
		URI	Parser
r1	t3	url=http://news.163.com	title=今日头条
	t2	host=163.com	
	t1		
r2	t5	url=http://www.163.com	content=科技新闻...
	t4	host=163.com	

- ① Row Key: 行键, Table 的主键, Table 中的记录按照 Row Key 排序。
- ② Timestamp: 时间戳, 每次数据操作对应的时间戳, 可以看成数据的版本号。
- ③ Column Family: 列簇, Table 在水平方向由一个或者多个 Column Family 组成, 一个 Column Family 中可以由任意多个 Column 组成, 即 Column Family 支持动态扩展, 无须预先定义 Column 的数量以及类型, 所有 Column 均以二进制格式存储, 用户需要自行进行类型转换。

(2) Table 和 Region

当 Table 随着记录数不断增加而变大后, 会逐渐分裂成多份 Split, 成为 region, 一个 Region 由[startkey,endkey)表示, 不同的 Region 会被 Master 分配给相应的 RegionServer 进行管理 (图 6.5)。

(3) Region 定位

HBase 如何找到某个 Row key (或者某个 Row Key 的 Range) 所在的 Region 呢? HBase 使用三层类似 B+树的结构来保存 Region 位置 (图 6.6)。

第一层: ZooKeeper 保存了 -ROOT- 表的位置。

第二层: -ROOT-表保存了 .META. 表所有 Region 的位置, 通过 -ROOT- 表, 可以访问 .META. 表的数据。

第三层: .META. 是一个特殊的表, 保存了 HBase 中所有数据表的 Region 位置信息。

需要注意的是:

- ① -ROOT- 表永远不会被 Split, 保证了最多需要三次跳转, 就能定位到任意 region。
- ② .META. 表每行保存一个 Region 的位置信息, Row Key 采用表名+表的最后一行编码。
- ③ 为了加快访问, .META. 表的全部 Region 都保存在内存中, Client 会将查询过的位置信息缓存起来, 缓存不会主动失效。

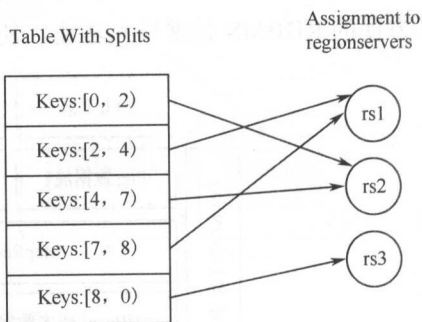


图 6.5 表的 Region 管理

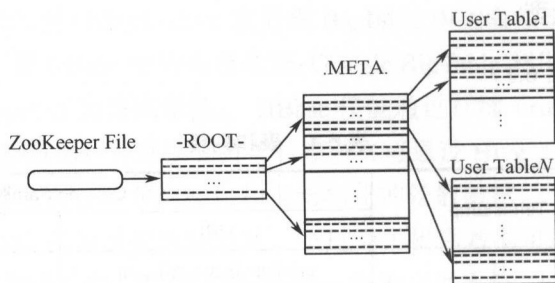


图 6.6 客户端访问 HBase 的流程

从图 6.6 中可以看出, Client 访问用户数据之前需要首先访问 ZooKeeper, 然后访问

-ROOT-表,接着访问.META.表,最后才能找到用户数据的位置去访问,中间需要多次网络操作。

2. HBase 访问接口

HBase 的访问通常有以下几种方式:

- ① 本地 Java 接口,最常规和高效的访问方式,适合 Hadoop MapReduce Job 并行批处理 HBase 表数据。
- ② HBase Shell, HBase 的命令行工具,最简单的接口,适合 HBase 管理使用。
- ③ Thrift Gateway,利用 Thrift 序列化技术,支持 C++、PHP、Python 等多种语言,适合其他异构系统在线访问 HBase 表数据。
- ④ REST Gateway,支持 REST 风格的 HTTP API 访问 HBase,解除了语言限制。
- ⑤ Pig,可以使用 Pig Latin 流式编程语言来操作 HBase 中的数据,和 Hive 类似,本质最终也是编译成 MapReduce Job 来处理 HBase 表数据,适合做数据统计。
- ⑥ Hive,通过 Hive,可以使用类似 SQL 语言来访问 HBase。

3. MapReduce On HBase

从图 6.4 中描述的 Hadoop 生态系统可以看到,MapRedcue 调度框架运行于 HBase 系统之上,通过两者的结合,使得 Hadoop 平台的大数据批处理能力得到淋漓尽致的体现。数据处理的交互过程如图 6.7 所示。

HBase 表和域的关系,可以类比 HDFS 文件和块的关系,域的集合构成了表。HBase 提供了配套的 TableInputFormat 和 TableOutputFormat API,可以方便地将 HBase 表作为 Hadoop MapReduce 的 Source 和 Sink,对于 MapReduce Job 应用开发人员来说,基本不需要关注 HBase 系统自身的细节。

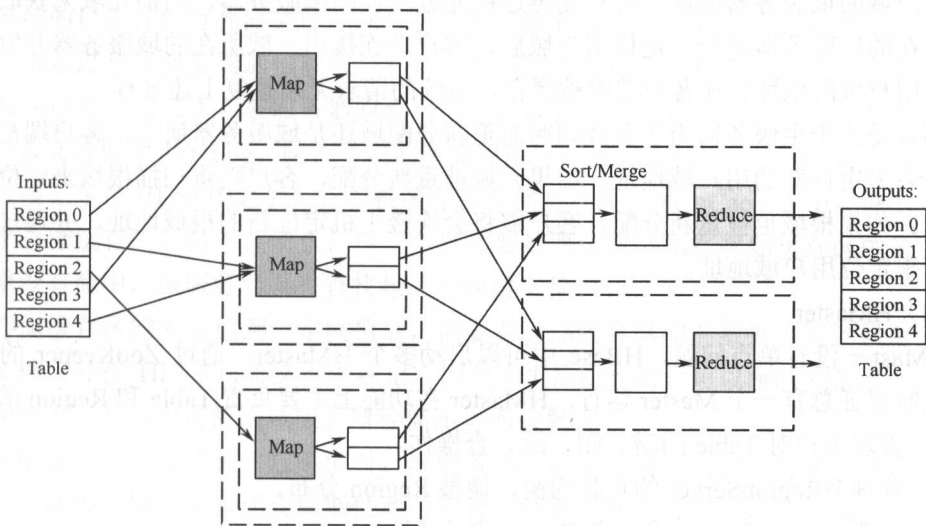


图 6.7 MapReduce 与 HBase 的交互流程图

4. HBase 系统架构

下面对图 6.8 中各个模块简单介绍一下。

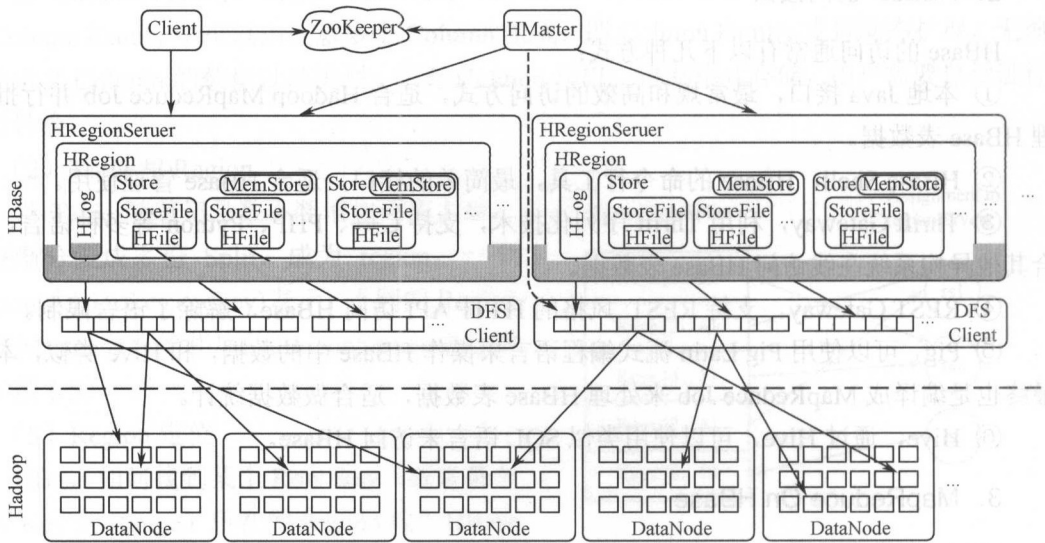


图 6.8 HBase 系统架构图

(1) HBase Client

HBase 客户端负责查找用户域所在的域服务器地址。HBase 客户端会与 HBase 主机交换消息以查找根域的位置，这是两者之间唯一的交流。

定位根域后，客户端连接根域所在的域服务器，并扫描根域获取元域信息，元域包含所需用户域的域服务器地址。客户端再连接元域所在的域服务器，扫描元域来获取所需用户域所在的域服务器地址。定位用户域后，客户端连接用户域所在的域服务器并发出读写请求。用户域的地址将在客户端中被缓存，后续的请求无须重复上述过程。

不管是由于主服务器为了负载均衡而重新分配域还是域服务器崩溃，客户端都会重新扫描元表来定位新的用户域地址。如果元域被重新分配，客户端将扫描根域来定位新的元域地址。如果根域也被重新分配，客户端将会连接主机定位新的根域地址，并通过重复上述过程来定位用户域地址。

(2) HMaster

HMaster 没有单点问题，HBase 中可以启动多个 HMaster，通过 ZooKeeper 的 Master 选举机制保证总有一个 Master 运行，HMaster 在功能上主要负责 Table 和 Region 的管理。

- ① 管理用户对 Table 的增、删、改、查操作。
- ② 管理 HRegionServer 的负载均衡，调整 Region 分布。
- ③ 在 Region Split 后，负责新 Region 的分配。
- ④ 在 HRegionServer 停机后，负责失效 HRegionServer 上的 Region 迁移。

(3) HRegionServer

HRegionServer 主要负责响应用户 I/O 请求, 向 HDFS 文件系统中读写数据, 是 HBase 中最核心的模块 (图 6.9)。

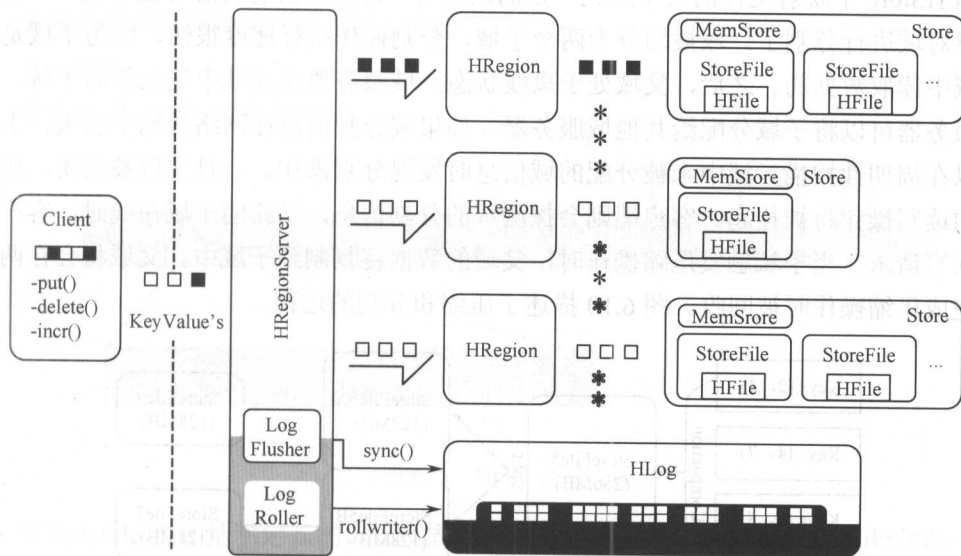


图 6.9 HRegionServer 架构图

HBase 域服务器主要有服务于主服务器分配的域、处理客户端的读写请求、缓冲区回写、压缩和分割域等功能。

每个域只能由一台域服务器来服务。当它开始服务于某域时, 它会从 HDFS 文件系统中读取该域的日志和所有存储文件。同时它还会管理 HDFS 文件的持久性存储。

客户端通过与主服务器通信获取域和域所在域服务器的列表信息后, 就可以直接向域服务器发送域读写请求了。域服务器收到写请求时, 首先将写请求信息写入一个预写日志文件中, 该文件取名为 HLog。同一个域的所有写请求都被记录在同一个 HLog 文件中。一旦写请求被记录在 HLog 中之后, 它将被缓存在存储缓存区 (MemCache) 中。每个 HStore 对应一个存储缓存区。对于读请求, 域服务器先要检测请求数据在存储缓存区中是否被命中, 如果没有命中, 域服务器再去查找相关的映射文件。

当存储缓存区的大小达到一定阈值后, 需要将存储缓存区中的数据回写到磁盘上, 形成映射文件, 并在 HLog 日志文件中标记。因此当再次执行时, 可以跳跃到最后一次回写之前的操作上。回写也可能因域服务器存储压力而被触发。

当映射文件的数量达到一定阈值时, 域服务器会将最近常写入的映射文件进行轻度的合并压缩。此外, 域服务器还会周期性地对所有的映射文件进行压缩, 使其成为单一的映射文件。之所以周期性地压缩所有的映射文件, 是因为最早的映射文件通常都比较大, 而最近的映射文件则要小很多, 压缩要消耗很多的时间, 具体消耗的时间主要取决于读取、

合并和写出最大映射文件所需要的 I/O 操作次数。压缩和处理读写请求是同时进行的。在一个新的映射文件移入之前，读写操作将被挂起，直到映射文件被加入 HStore 的活跃映射文件列表中，且已合并的旧映射文件被删除后，才会释放读写操作。

当 HStore 中映射文件的大小达到一定的阈值时（目前默认的阈值为 256MB），域服务器就要对域进行分割了。域被均分为两个子域，分割操作执行速度很快，因为子域是直接由父域中读取数据的。之后，父域处于离线状态。域服务器在元域中记录新的子域，并通知主服务器可以将子域分配给其他域服务器。如果域分割消息在网络传输中丢失，主服务器可以在周期性扫描元域中未被分配的域信息时发现分割操作。一旦父域被关闭，所有对父域的读写操作将被挂起。客户端则会探测域的分割信息，当新的子域在线时，客户端再发出读写请求。当子域触发压缩操作时，父域的数据将复制到子域中。父域将会在两个子域都完成压缩操作时被回收。图 6.10 描述了压缩和分割的过程。

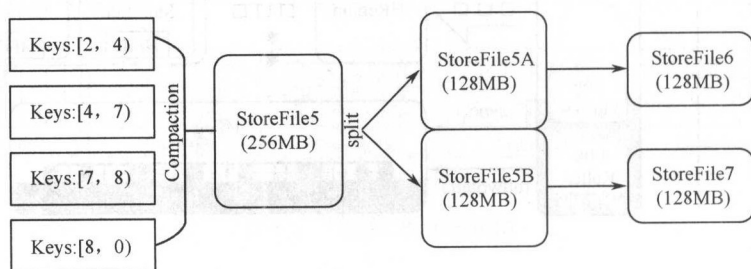


图 6.10 压缩和分割过程

在理解了上述 HStore 的基本原理后，还必须了解一下 HLog 的功能，因为上述的 HStore 在系统正常工作的前提下是没有问题的，但是在分布式系统环境中，无法避免系统出错或者宕机，因此一旦 HRegionServer 意外退出，MemStore 中的内存数据将会丢失，这就需要引入 HLog 了。

每个 HRegionServer 中都有一个 HLog 对象，HLog 是一个实现 Write Ahead Log 的类，在每次用户操作写入 MemStore 的同时，也会写一份数据到 HLog 文件中（HLog 文件格式见后续），HLog 文件定期会滚动出新的，并删除旧的文件（已持久化到 StoreFile 中的数据）。当 HRegionServer 意外终止后，HMaster 会通过 ZooKeeper 感知到，HMaster 首先会处理遗留的 HLog 文件，将其中不同 Region 的 Log 数据进行拆分，分别放到相应 Region 的目录下，然后再将失效的 Region 重新分配，领取到这些 Region 的 HRegionServer 在 Load Region 的过程中，会发现有历史 HLog 需要处理，因此会 Replay HLog 中的数据到 MemStore 中，然后 Flush 到 StoreFiles，完成数据恢复。

5. HBase 存储格式

HBase 中的所有数据文件都存储在 Hadoop HDFS 文件系统中，主要包括上面提出的两种文件类型。

① HFile, HBase 中 KeyValue 数据的存储格式, HFile 是 Hadoop 的二进制格式文件, 实际上 StoreFile 就是对 HFile 做了轻量级包装, 即 StoreFile 底层就是 HFile。

② HLog File, HBase 中 WAL (Write Ahead Log) 的存储格式, 物理上是 Hadoop 的 Sequence File。

下面分别对这两种文件类型进行介绍。

(1) HFile

图 6.11 是 HFile 的存储格式。

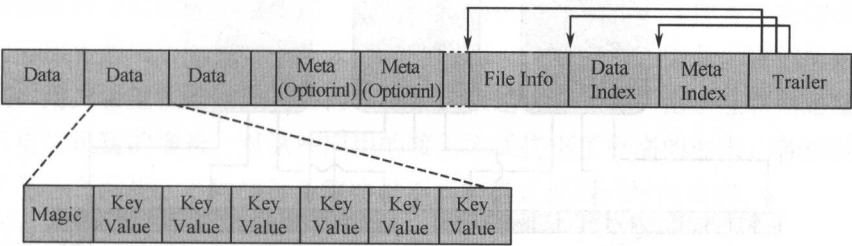


图 6.11 HFile 存储格式

首先 HFile 文件是不定长的, 长度固定的只有其中的两块: Trailer 和 File Info。正如图中所示, Trailer 中有指针指向其他数据块的起始点。File Info 中记录了文件的一些 Meta 信息, 例如 AVG_KEY_LEN、AVG_VALUE_LEN、LAST_KEY、COMPARATOR、MAX_SEQ_ID_KEY 等。Data Index 和 Meta Index 块记录了每个 Data 块和 Meta 块的起始点。

Data Block 是 HBase I/O 的基本单元, 为了提高效率, HRegionServer 中有基于 LRU 的 Block Cache 机制。每个 Data 块的大小可以在创建一个 Table 的时候通过参数指定, 大号的 Block 有利于顺序 Scan, 小号的 Block 利于随机查询。每个 Data 块除了开头的 Magic 以外就是一个 Key Value 对拼接而成的, Magic 内容就是一些随机数字, 目的是防止数据损坏。后面会详细介绍每个 Key Value 对的内部构造。

HFile 里面的每个 Key Value 对就是一个简单的 byte 数组。但是这个 byte 数组里面包含了很多项, 并且有固定的结构 (图 6.12)。

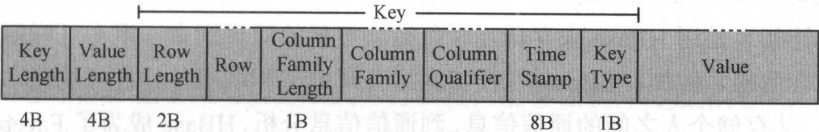


图 6.12 HFile 数据项结构

开始是两个固定长度的数值, 分别表示 Key 的长度和 Value 的长度。紧接着是 Key, 开始是固定长度的数值, 表示 RowKey 的长度, 紧接着是 RowKey, 然后是固定长度的数值, 表示 Family 的长度, 然后是 Family, 接着是 Qualifier, 然后是两个固定长度的数值, 表示 Time Stamp 和 Key Type (Put/Delete)。Value 部分没有这么复杂的结构, 就是纯粹的

二进制数据。

(2) HLog File

图 6.13 为 HLog 文件的结构，其实 HLog 文件就是一个普通的 Hadoop Sequence File，Sequence File 的 Key 是 HLogKey 对象，HLogKey 中记录了写入数据的归属信息，除了 Table 和 Region 名字外，同时还包括 Sequence Number 和 Timestamp，Timestamp 是写入时间，Sequence Number 的起始值为 0，或者是最近一次存入文件系统的 Sequence Number。

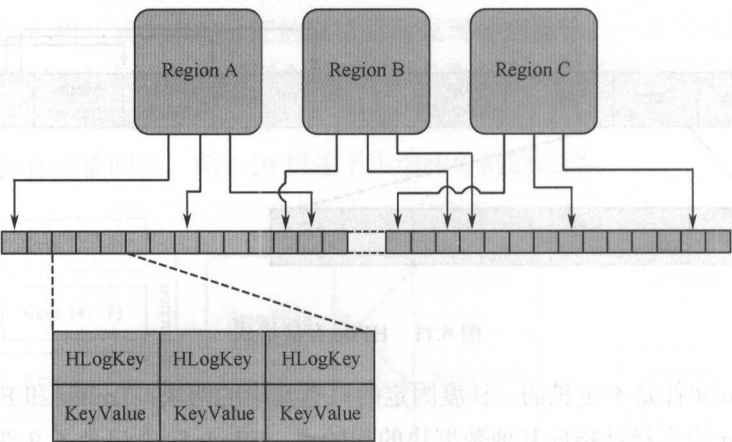


图 6.13 HLog 文件结构图

HLog Sequence File 的 Value 是 HBase 的 KeyValue 对象，即对应 HFile 中的 KeyValue。

6. HBase 使用场景

HBase 被证实是一个强大的工具，尤其是在已经使用 Hadoop 的场合。其在“婴儿期”时，就被快速部署到其他公司的生产环境中，并得到了开发人员的支持。如今，HBase 已经是 Apache 顶级项目，有着众多的开发人员和兴旺的用户社区。它成为了一个核心的基础架构部件，运行在世界上许多公司（如 StumbleUpon、Trend Micro、Facebook、Twitter、Salesforce 和 Adobe）的大规模生产环境中。

一个设计初衷是用 HBase 来存储互联网持续更新的网页副本，用在互联网相关的其他方面也是很合适的。例如，HBase 在社交网络公司内部和周围各种各样的需求中，也找到了用武之地。从存储个人之间的通信信息，到通信信息分析，HBase 成为了 Facebook、Twitter 和 StumbleUpon 等公司的关键基础设施。在这个领域，HBase 有 3 种主要使用场景，但不限于这 3 种。接下来我们将介绍这 3 种主要的使用场景：

- ① 抓取增量数据；
- ② 内容服务；
- ③ 信息交换。

(1) 抓取增量数据

数据通常是细水长流的，不断累加到已有的数据库中以备将来使用，如分析、处理和服务。许多 HBase 使用场景属于这一类——使用 HBase 作为数据存储，抓取来自各种数据源的增量数据。例如，这种数据源可能是网页爬虫（我们讨论过的 BigTable 典型问题），可能是记录用户看了什么广告和看了多长时间的广告效果数据，也可能是记录各种参数的时间序列数据。

(2) 内容服务

传统数据库最主要的使用场合之一是为用户提供内容服务。各种各样的数据库支撑着提供各种内容服务的应用系统。多年来，这些应用一直在发展，因此，它们所依赖的数据库也在发展。用户希望使用和交互的内容种类越来越多。此外，由于互联网迅猛的增长以及终端设备更加迅猛的增长，对这些应用的接入方式提出了更高的要求。各种各样的终端设备带来了另一个挑战：不同的设备需要以不同的格式使用同样的内容。

(3) 信息交换

各种社交网络破土而出，世界变得越来越小。社交网站的一个重要作用就是帮助人们进行互动。有时互动在群组内发生（小规模和大规模），有时互动在两个人之间发生。想想看，数亿人通过社交网络进行对话的场面。单单和远处的人对话还不足以让人满意，人们还想看看和其他人对话的历史记录。让社交网络公司感到幸运的是，保存这些历史记录很廉价，大数据领域的创新可以帮助他们充分利用廉价的存储。

6.2 MapReduce

MapReduce 是一种用于大规模数据集（大于 1TB）的并行运算的编程模型。概念 Map（映射）和 Reduce（归约）以及它们的主要思想，都是从函数式编程语言里借用而来的，同时也包含了从矢量编程语言里借来的特性。MapReduce 极大地方便了编程人员在不会分布式并行编程的情况下，将自己的程序运行在分布式系统上。

许多人认为这种编程方式的重大变化将带来一次软件的并发危机，因为传统的软件方式基本上是单指令单数据流的顺序执行，这种顺序执行十分符合人类的思考习惯，却与并发并行编程格格不入。基于集群的分布式并行编程，能够让软件与数据同时运行在连成一个网络的许多台计算机上，这里的每一台计算机均可以是一台普通的 PC。这样的分布式并行环境的最大优点是，可以很容易地通过增加计算机来扩充新的计算节点，并由此获得不可思议的海量计算能力，同时又具有相当强的容错能力，一批计算节点失效也不会影响计算的正常进行以及结果的正确性。Google 就是这么做的，他们使用了叫做 MapReduce 的并行编程模型进行分布式并行编程，运行在叫做 GFS（Google File System）的分布式文件系统上，为全球亿万用户提供搜索服务。

Hadoop 实现了 Google 的 MapReduce 编程模型，提供了简单易用的编程接口，也提供了它自己的分布式文件系统 HDFS，与 Google 不同的是，Hadoop 是开源的，任何人都可以使用这个框架来进行并行编程。如果说分布式并行编程的难度足以让普通程序员望而生畏的话，开源的 Hadoop 的出现，则极大地降低了它的门槛。你会发现，基于 Hadoop 编程非常简单，无须任何并行开发经验，你也可以轻松地开发出分布式的并程序，并让其令人难以置信地同时运行在数百台机器上，然后在短时间内完成海量数据的计算。你可能会觉得你不可能拥有数百台机器来运行你的并程序，而事实上，随着云计算的普及，任何人都可以轻松获得这样的海量计算能力。例如，现在 Amazon 公司的云计算平台 Amazon EC2 已经提供了这种按需计算的租用服务。

6.2.1 第一代 MapReduce (MRv1)

MapReduce 是 Google 提出的一种并行化编程模型，简而言之，它体现了分而治之的策略，它将复杂的、运行于大规模集群上的并行计算过程高度地抽象为两个函数：Map 和 Reduce。适合用 MapReduce 来处理的数据集（或任务），需要满足一个基本要求：待处理的数据集可以分解成许多小的数据集，而且每一个小数据集都可以完全并行地进行处理。一个 MapReduce 作业（job）通常会把输入的数据集切分为若干独立的数据块，由 Map 任务（task）以完全并行的方式处理它们。框架会对 Map 的输出先进行排序，然后把结果输入 Reduce 任务。通常作业的输入和输出都会被存储在文件系统中。整个框架负责任务的调度和监控，以及重新执行已经失败的任务。通常，MapReduce 框架和分布式文件系统是运行在一组相同的节点上的，也就是说，计算节点和存储节点通常在一起。这种配置允许框架在那些已经存好数据的节点上高效地调度任务，这可以使整个集群的网络带宽被非常高效地利用。MapReduce 框架由单独一个 Master JobTracker 和每个集群节点一个 Slave TaskTracker 共同组成。这个 Master 负责调度构成一个作业的所有任务，这些任务分布在不同的 Slave 上，Master 监控它们的执行，重新执行已经失败的任务。而 Slave 仅负责执行由 Master 指派的任务。应用程序至少应该指明输入/输出的位置（路径），并通过实现合适的接口或抽象类提供 Map 和 Reduce 函数，再加上其他作业的参数，就构成了作业配置（job configuration）。然后，Hadoop 的 Job Client 提交作业（jar 包/可执行程序等）和配置信息给 JobTracker，后者负责分发这些软件和配置信息给 Slave，调度任务且监控它们的执行，同时提供状态和诊断信息给 Job Client。虽然 Hadoop 框架是用 Java 实现的，但 MapReduce 应用程序则不一定要用 Java 来写。

一张图胜过千言万语，下面结合图 6.14 来介绍一下 Hadoop MapReduce 框架的执行原理。

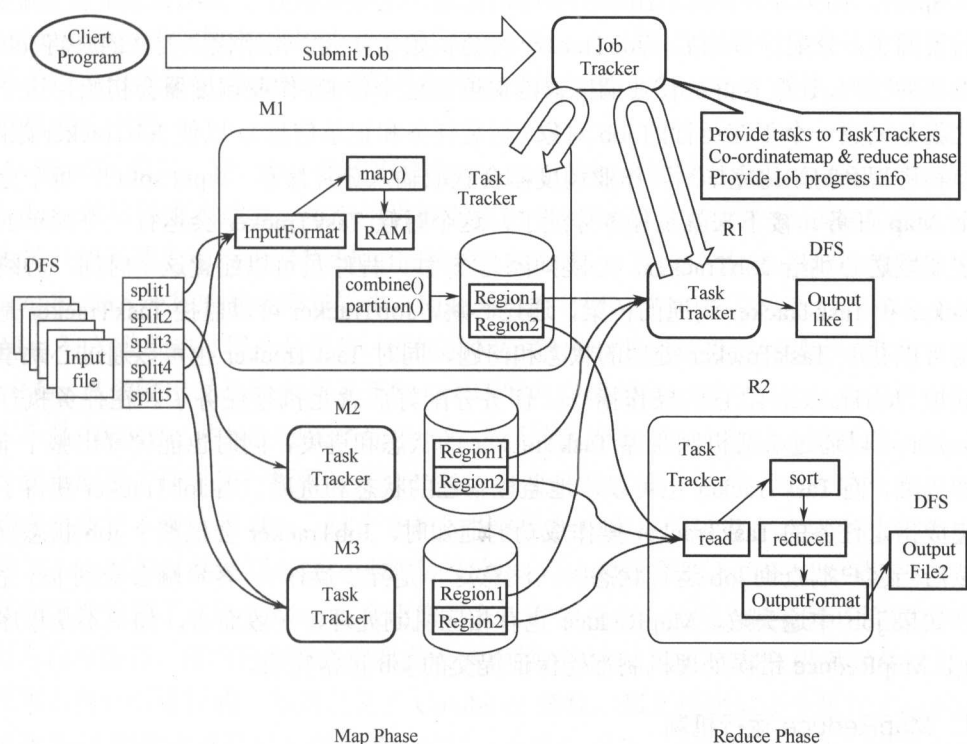


图 6.14 MapReduce 框架原理图

MapReduce 运行机制可以从很多不同的角度来描述, 比如说从 MapReduce 运行流程来讲解, 也可以从计算模型的逻辑流程来讲解。

首先讲讲物理实体, MapReduce 作业的执行涉及四个独立的实体。

① 客户端 (client): 编写 MapReduce 程序, 配置作业, 提交作业, 这就是程序员完成的工作。

② JobTracker: 初始化作业, 分配作业, 与 TaskTracker 通信, 协调整个作业的执行。

③ TaskTracker: 保持与 JobTracker 的通信, 在分配的数据片段上执行 Map 或 Reduce 任务, TaskTracker 和 JobTracker 有一个很重要的区别, 就是在执行任务时 TaskTracker 可以有多个, 而 JobTracker 则只有一个。

④ HDFS: 保存作业的数据、配置信息等, 最后的结果也保存在 HDFS 上。

那么 MapReduce 到底是如何运行的呢?

首先是客户端要编写好 MapReduce 程序, 配置好 MapReduce 的作业 (job), 接下来就是提交 job 到 JobTracker 上, 这个时候 JobTracker 就会构建这个 job, 具体就是分配一个新的 job 任务的 ID 值, 接下来它会做检查操作, 这个检查就是确定输出目录是否存在, 如果存在那么 job 就不能正常运行下去, JobTracker 会抛出错误给客户端, 接下来还要检查输入目录是否存在, 如果不存在同样抛出错误, 如果存在 JobTracker 会根据输入计算输入分片

(Input Split), 如果分片计算不出来也会抛出错误, 这些都做好了 JobTracker 就会配置 job 需要的资源了。分配好资源后, JobTracker 就会初始化作业, 初始化主要做的是将 job 放入一个内部的队列, 让配置好的作业调度器能调度到这个作业, 作业调度器会初始化这个 job, 初始化就是创建一个正在运行的 job 对象(封装任务和记录信息), 以便 JobTracker 跟踪 job 的状态和进程。初始化完毕后, 作业调度器会获取输入分片信息(input split), 每个分片创建一个 Map 任务。接下来就是任务分配了, 这个时候 TaskTracker 会运行一个简单的循环机制定期发送心跳给 JobTracker, 心跳间隔是 5 秒, 程序员可以配置这个时间, 心跳就是 JobTracker 和 TaskTracker 沟通的桥梁, 通过心跳, JobTracker 可以监控 TaskTracker 是否存活, 也可以获取 TaskTracker 处理的状态和问题, 同时 TaskTracker 也可以通过心跳里的返回值获取 JobTracker 给它的操作指令。任务分配好后就是执行任务了。在任务执行时, JobTracker 可以通过心跳机制监控 TaskTracker 的状态和进度, 同时也能计算出整个 job 的状态和进度, 而 TaskTracker 也可以本地监控自己的状态和进度。当 JobTracker 获得了最后一个完成指定任务的 TaskTracker 操作成功的通知时, JobTracker 会把整个 job 状态置为成功, 然后当客户端查询 job 运行状态时(注意这个是异步操作), 客户端会查到 job 完成的通知。如果 job 中途失败, MapReduce 也有相应机制处理, 一般而言, 如果不是程序本身有 bug, MapReduce 错误处理机制都能保证提交的 job 正常完成。

1. MapReduce 运行机制

下面从逻辑实体的角度讲解 MapReduce 运行机制, 这些按照时间顺序包括: 输入分片(input split)、Map 阶段、Combiner 阶段、Shuffle 阶段和 Reduce 阶段。

① 输入分片(input split): 在进行 Map 计算之前, MapReduce 会根据输入文件计算输入分片, 每个输入分片针对一个 Map 任务, 输入分片存储的并非数据本身, 而是一个分片长度和一个记录数据的位置的数组, 输入分片往往和 HDFS 的 block(块)关系很密切, 假如设定 HDFS 的块的大小是 64MB, 如果输入有三个文件, 大小分别是 3MB、65MB 和 127MB, 那么 MapReduce 会把 3MB 文件分为一个输入分片, 65MB 则是两个输入分片, 而 127MB 也是两个输入分片, 换句话说, 如果在 Map 计算前做输入分片调整, 例如合并小文件, 那么就会有 5 个 Map 任务将执行, 而且每个 Map 执行的数据大小不均, 这也是 MapReduce 优化计算的一个关键点。

② Map 阶段: 就是程序员编写好 Map 函数了, 因此 Map 函数效率相对好控制, 而且一般 Map 操作都是本地化操作, 也就是在数据存储节点上进行。

③ Combiner 阶段: Combiner 阶段是程序员可以选择的, Combiner 其实也是一种 Reduce 操作。Combiner 是一个本地化的 Reduce 操作, 它是 Map 运算的后续操作, 主要是在 Map 计算出中间文件前做一个简单的合并重复 key 值的操作, 例如我们对文件里的单词频率做统计, Map 计算时如果碰到一个 Hadoop 的单词就会记录为 1, 但是这篇文章里 Hadoop 可能会出现 n 次, 那么 Map 输出文件冗余就会很多, 因此在 Reduce 计算前对相同的 key 做一个合并操作, 那么文件会变小, 这样就提高了宽带的传输速率, 毕竟 Hadoop 宽带资源往

往是计算的瓶颈，也是最为宝贵的资源，但是 Combiner 操作是有风险的，使用它的原则是 Combiner 的输出不会影响到 Reduce 计算的最终输入，例如，如果计算只是求总数，最大值、最小值，可以使用 Combiner，但是做平均值计算使用 Combiner 的话，最终的 Reduce 计算结果就会出错。

④ Shuffle 阶段：将 Map 的输出作为 Reduce 的输入的过程就是 Shuffle 了，这个是 MapReduce 优化的重点地方。本节不讲怎么优化 Shuffle 阶段，讲讲 Shuffle 阶段的原理，因为大部分的书籍里都没讲清楚 Shuffle 阶段。Shuffle 一开始就是 Map 阶段做输出操作，一般 MapReduce 计算的都是海量数据，Map 输出时不可能把所有文件都放到内存中操作，因此 Map 写入磁盘的过程十分复杂，更何况 Map 输出时还要对结果进行排序，内存开销是很大的，Map 在做输出时会在内存里开启一个环形内存缓冲区，这个缓冲区专门用来输出，默认大小是 100MB，并且在配置文件里为这个缓冲区设定了一个阈值，默认是 0.80（这个大小可以在配置文件里进行配置），同时 Map 还会为输出操作启动一个守护线程，如果缓冲区的内存达到了阈值的 80%，这个守护线程就会把内容写到磁盘上，这个过程叫 Spill，另外的 20% 内存可以继续写入要写进磁盘的数据，写入磁盘和写入内存操作是互不干扰的，如果缓存区被填满了，那么 Map 就会阻塞写入内存的操作，让写入磁盘操作完成后再继续执行写入内存操作，前面讲到写入磁盘前会有个排序操作，这是在写入磁盘操作时进行的，不是在写入内存时进行的，如果定义了 Combiner 函数，那么排序前还会执行 Combiner 操作。每次 Spill 操作也就是写入磁盘操作时写一个溢出文件，也就是说，Map 输出有几次，Spill 就会产生多少个溢出文件，等 Map 输出全部做完后，Map 会合并这些输出文件。这个过程里还会有一个 Partitioner 操作，对于这个操作很多人都很迷糊，其实 Partitioner 操作和 Map 阶段的输入分片（Input split）很像，一个 Partitioner 对应一个 Reduce 作业，如果 MapReduce 操作只有一个 Reduce 操作，那么 Partitioner 就只有一个，如果有多个 Reduce 操作，那么 Partitioner 就会有多个，Partitioner 就是 Reduce 的输入分片，这个程序员可以编程控制，这是提高 Reduce 效率的一个关键所在。到了 Reduce 阶段就是合并 Map 输出文件了，Partitioner 会找到对应的 Map 输出文件，然后进行复制操作，复制操作时 Reduce 会开启几个复制线程，这些线程默认个数是 5 个，程序员也可以在配置文件里更改复制线程的个数，这个复制过程和 Map 写入磁盘过程类似，也有阈值和内存大小，阈值一样可以在配置文件里配置，而内存大小直接使用 Reduce 的 TaskTracker 的内存大小，复制时 Reduce 还会进行排序操作和合并文件操作，这些操作完了就会进行 Reduce 计算了。

⑤ Reduce 阶段：和 Map 函数一样也是程序员编写的，最终结果是存储在 HDFS 上的。

2. MapReduce 实例分析：“HelloWorld”

Hadoop WordCount 的例子就如同编程语言中的“HelloWorld”程序一样经典，是 MapReduce 的入门程序。WordCount 顾名思义就是需要计算出文件中各个单词的频数，要求输出结果按照单词的字母顺序进行排序，每个单词和其频数占一行，单词和频数之间有间隔。比如，输入一个文件，其内容如下：

```
hello world
hello hadoop
hello mapreduce
```

对应上面给出的输入样例，其输出样例为：

```
hadoop 1
hello 3
mapreduce 1
world 1
```

(1) WordCount 的设计思路

上面这个应用实例的解决方案很直接，就是将文件内容切分成单词，然后将所有相同的单词聚集到一起，最后，计算单词出现的次数进行输出。针对 MapReduce 并程序计原则可知，解决方案中的内容切分步骤和数据不相关，可以并行化处理，每个拿到原始数据的机器只要将输入数据切分成单词就可以了。所以，可以在 Map 阶段完成单词切分的任务。另外，相同单词的频数计算也可以并行化处理。根据实例要求来看，不同单词之间的频数不相关，所以，可以将相同的单词交给一台机器来计算频数，然后输出最终结果。这个过程可以交给 Reduce 阶段完成。至于将中间结果根据不同单词进行分组后再发送给 Reduce 机器，这正好是 MapReduce 过程中的 Shuffle 能够完成的。至此，这个实例的 MapReduce 程序就设计出来了。Map 阶段完成由输入数据到单词切分的工作，Shuffle 阶段完成相同单词的聚集和分发工作（这个过程是 MapReduce 的默认过程，不用具体配置），Reduce 阶段完成接收所有单词并计算其频数的工作。由于 MapReduce 中传递的数据都是 <key, value>形式的，并且 Shuffle 排序聚集分发是按照 key 值进行的，所以，将 Map 的输出设计成由 word 作为 key，1 作为 value 的形式，它表示单词出现了 1 次（Map 的输入采用 Hadoop 默认的输入方式，即文件的一行作为 value，行号作为 key）。Reduce 的输入是 Map 输出聚集后的结果，即 <key, value-list>，具体到这个实例就是 <word, {1,1,1,1,...}>，Reduce 的输出会设计成与 Map 输出相同的形式，只是后面的数值不再是固定的 1，而是具体算出的 word 所对应的频数。

(2) WordCount 代码

采用 MapReduce 进行词频统计的 WordCount 代码如下：

```
public class WordCount {
    public static class TokenizerMapper extends Mapper<Object, Text, Text, IntWritable>
    {
        private final static IntWritable one = new IntWritable(1);
        private Text word = new Text();
        public void map(Object key, Text value, Context context ) throws IOException,
        InterruptedException
```



```

        {
            StringTokenizer itr = new StringTokenizer(value.toString());
            while (itr.hasMoreTokens())
            {
                word.set(itr.nextToken()); context.write(word, one);
            }
        }
    }

    public static class IntSumReducer extends Reducer<Text, IntWritable, Text, IntWritable>
    {
        private IntWritable result = new IntWritable();

        public void reduce(Text key, Iterable<IntWritable> values, Context context) throws IOException,
        InterruptedException
        {
            int sum = 0;
            for (IntWritable val : values)
            {
                sum += val.get();
            }
            result.set(sum);
            context.write(key, result);
        }
    }

    public static void main(String[] args) throws Exception
    {
        Configuration conf = new Configuration();
        String[] otherArgs = new GenericOptionsParser(conf, args).getRemainingArgs();
        if (otherArgs.length != 2)
        {
            System.err.println("Usage: wordcount <in> <out>");
            System.exit(2);
        }

        Job job = new Job(conf, "word count");
        job.setJarByClass(WordCount.class);
        job.setMapperClass(TokenizerMapper.class);
        job.setCombinerClass(IntSumReducer.class);
    }

```



```

        job.setReducerClass(IntSumReducer.class);
        job.setOutputKeyClass(Text.class);
        job.setOutputValueClass(IntWritable.class);
        FileInputFormat.addInputPath(job, new Path(otherArgs[0]));
        FileOutputFormat.setOutputPath(job, new Path(otherArgs[1]));
        System.exit(job.waitForCompletion(true) ? 0 : 1);
    }
}

```

(3) 过程解释

Map 操作的输入是<Key, Value>形式的，其中，Key 是文档中某行的行号，Value 是该行的内容。Map 操作会将输入文档中每一个单词的出现输出到中间文件，如图 6.15 所示。

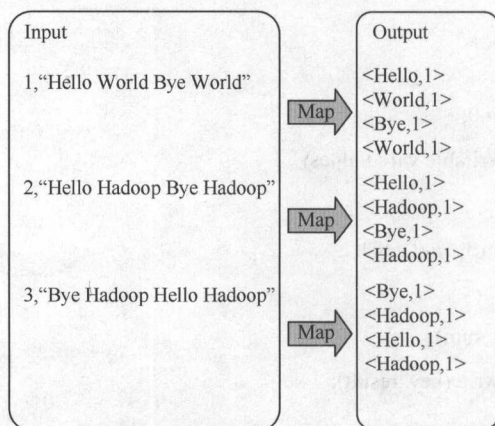


图 6.15 Map 过程示意图

Reduce 操作的输入是单词和出现次数的序列（图 6.16）。用上面的例子来说，就是<"Hello", [1,1,1]>, <"World", [1,1]>, <"Bye", [1,1,1]>, <"Hadoop", [1,1,1,1]>等。然后根据每个单词，算出总的出现次数。最后输出排序后的最终结果：<"Bye", 3>, <"Hadoop", 4>, <"Hello", 3>, <"World", 2>。

整个 MapReduce 过程实际的执行顺序如下。

① MapReduce Library 将 Input 分成 M 份，这里的 Input Splitter 也可以是多台机器并行 Split。

② Master 将 M 份 Job 分给空闲状态的 M 个 worker 来处理。

③ 对于输入中的每一个<Key, Value> 进行 Map 操作，将中间结果缓冲在内存里。

④ 定期地（或者根据内存状态）将缓冲区中的中间信息刷写到本地磁盘上，并且把文件信息传回给 Master（Master 需要把这些信息发送给 Reduce worker）。这里最重要的一点是，在写磁盘的时候，需要将中间文件做 Partition（比如 R 个）。拿上面的例子来说，如果

把所有的信息存到一个文件, Reduce worker 又会变成瓶颈。只需要保证相同 Key 能出现在同一个 Partition 里就可以把这个问题分解。

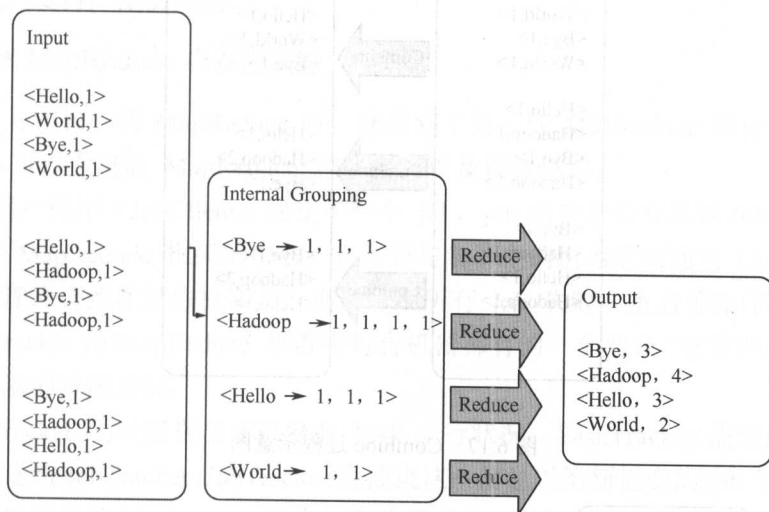


图 6.16 Reduce 过程示意图

⑤ R 个 Reduce worker 开始工作, 从不同的 Map worker 的 Partition 那里拿到数据, 用 key 进行排序 (如果内存中放不下需要用到外部排序)。很显然, 排序 (或者说 Group) 是 Reduce 函数之前必须做的一步。这里很关键的是, 每个 Reduce worker 会从很多 Map worker 那里拿到 $X(0 < X < R)$ Partition 的中间结果, 这样, 所有属于这个 key 的信息已经都在这个 worker 上了。

⑥ Reduce worker 遍历中间数据, 对每一个唯一 Key, 执行 Reduce 函数 (参数是这个 Key 以及相对应的一系列 Value)。

⑦ 执行完毕后, 唤醒用户程序, 返回结果 (最后应该有 R 份 Output, 每个 Reduce worker 一个)。

可见, 这里的“分”(Divide)体现在两步, 分别是将输入分成 M 份, 以及将 Map 的中间结果分成 R 份。将输入分开通常很简单, 而把 Map 的中间结果分成 R 份, 通常用 $\text{hash}(\text{key}) \bmod R$ 这个结果作为标准, 保证相同的 Key 出现在同一个 Partition 里。当然, 使用者也可以指定自己的 Partition 函数, 比如, 对于 URL key, 如果希望同一个 Host 的 URL 出现在同一个 Partition, 可以用 $\text{hash}(\text{Hostname}(\text{urlkey})) \bmod R$ 作为 Partition 函数。另外, 对于上面的例子来说, 每个文档中都可能会出现成千上万的 $\langle \text{the}, 1 \rangle$ 这样的中间结果, 琐碎的中间文件必然导致传输上的开销。因此, MapReduce 还支持用户提供 Combiner 函数。这个函数通常与 Reduce 函数有相同的实现, 不同点在于 Reduce 函数的输出是最终结果, 而 Combiner 函数的输出是 Reduce 函数的输入。图 6.15 中的 Map 过程输出结果, 如果采用 Combiner 函数, 则可以得到如图 6.17 所示的输出, 这个输出结果可以作为 Reduce 过程的输入 (图 6.18)。

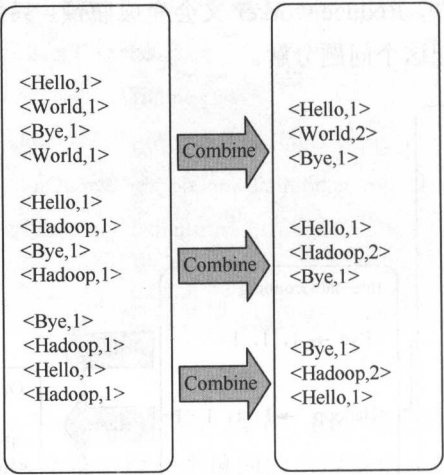


图 6.17 Combine 过程示意图

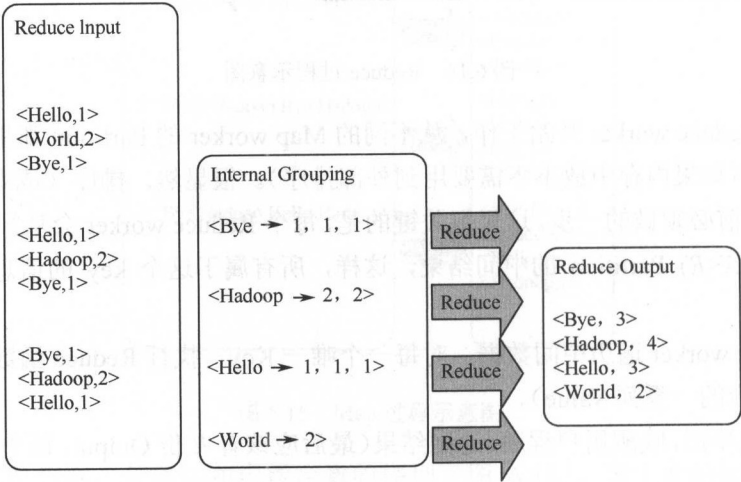


图 6.18 以 Combine 输出结果作为输入的 Reduce 过程示意图

6.2.2 MapReduce 2.0——Yarn

Yarn 带来了巨大的改变，改变了 Hadoop 计算组件（MapReduce）切分和重新组成处理任务的方式，因为 Yarn 把 MapReduce 的追踪组件切分成两个不同部分：资源管理器和应用调度。这样的数据管理工具，有助于更加轻松地同时运行 MapReduce 或 Storm 任务以及 HBase 等服务。Hadoop 共同创始人之一 Doug Cutting 表示：“它使得其他不是 MapReduce 的工作负载现在可以更有效地与 MapReduce 分享资源。现在这些系统可以动态地分享资源，资源也可以设置优先级。”

Cutting 和 Bhandarkar 都承认, 这种方法受到了 Apache 项目 Mesos 集群管理系统以及谷歌 Borg 和 Omega 秘密项目的一些影响。Yarn 的出现, 使得 Hadoop 变成一个针对数据中心的操作系统, 支持广泛的应用。

1. 第一代 MapReduce 存在的问题

在上一节介绍第一代 MapReduce 时, 就提到了第一代 MapReduce 设计上存在一些问题, 从图 6.14 可以看出原 MapReduce 程序的流程及设计思路。

① 首先用户程序 (JobClient) 提交了一个 job, job 的信息会发送到 Job Tracker 中, Job Tracker 是 MapReduce 框架的中心, 它需要与集群中的机器定时通信 (heartbeat), 需要管理哪些程序应该跑在哪些机器上, 需要管理所有 job 失败、重启等操作。

② TaskTracker 是 MapReduce 集群中每台机器都有一个部分, 它做的事情主要是监视自己所在机器的资源情况。

③ TaskTracker 同时监视当前机器的 Task 运行状况。TaskTracker 需要把这些信息通过 heartbeat 发送给 JobTracker, JobTracker 会搜集这些信息以给新提交的 job 分配运行机器。

可以看得出原来的 MapReduce 架构是简单明了的, 在最初推出的几年, 也取得了众多的成功案例, 获得了业界广泛的支持和肯定。但随着分布式系统集群的规模和其工作负荷的增长, 原框架的问题逐渐浮出水面, 主要的问题如下。

① JobTracker 是 MapReduce 的集中处理点, 存在单点故障。

② JobTracker 完成了太多的任务, 造成了过多的资源消耗, 当 MapReduce 作业非常多的时候, 会造成很大的内存开销, 也增加了 JobTracker 失败的风险, 业界普遍总结出 Hadoop 的 MapReduce 最多只能支持 4000 节点主机。

③ 在 TaskTracker 端, 以 Map/Reduce Task 的数目作为资源的表示过于简单, 没有考虑到 CPU/内存的占用情况, 如果两个大内存消耗的 Task 被调度到了一起, 很容易出现内存不足。

④ 在 TaskTracker 端, 把资源强制划分为 Map Task Slot 和 Reduce Task Slot, 如果系统中只有 Map Task 或者只有 Reduce Task, 会造成资源的浪费, 也就是前面提到的集群资源利用的问题。

⑤ 源代码层面分析时, 会发现代码非常难读, 常常因为一个 class 做了太多的事情, 代码量达 3000 多行, 造成 class 的任务不清晰, 增加 bug 修复和版本维护的难度。

⑥ 从操作的角度来看, 现在的 Hadoop MapReduce 框架在有任何重要的或者不重要的变化 (例如 bug 修复、性能提升和特性化) 时, 都会强制进行系统级别的升级更新。更糟的是, 它不管用户的喜好, 强制让分布式集群系统的每一个用户端同时更新。这些更新会让用户为了验证他们之前的应用程序是不是适用新的 Hadoop 版本而浪费大量时间。

2. Yarn 框架原理及运作机制

从业界使用分布式系统的变化趋势和 Hadoop 框架的长远发展来看, MapReduce 的 JobTracker/TaskTracker 机制需要大规模调整来修复它在可扩展性、内存消耗、线程模型、

可靠性和性能上的缺陷。在过去的几年中，Hadoop 开发团队做了一些 bug 的修复，但是最近这些修复的成本越来越高，这表明对原框架做出改变的难度越来越大。

为从根本上解决旧 MapReduce 框架的性能瓶颈，促进 Hadoop 框架的更长远发展，从 0.23.0 版本开始，Hadoop 的 MapReduce 框架完全重构，发生了根本的变化。新的 Hadoop MapReduce 框架命名为 MapReduceV2，或者叫 Yarn，其架构图如图 6.19 所示。

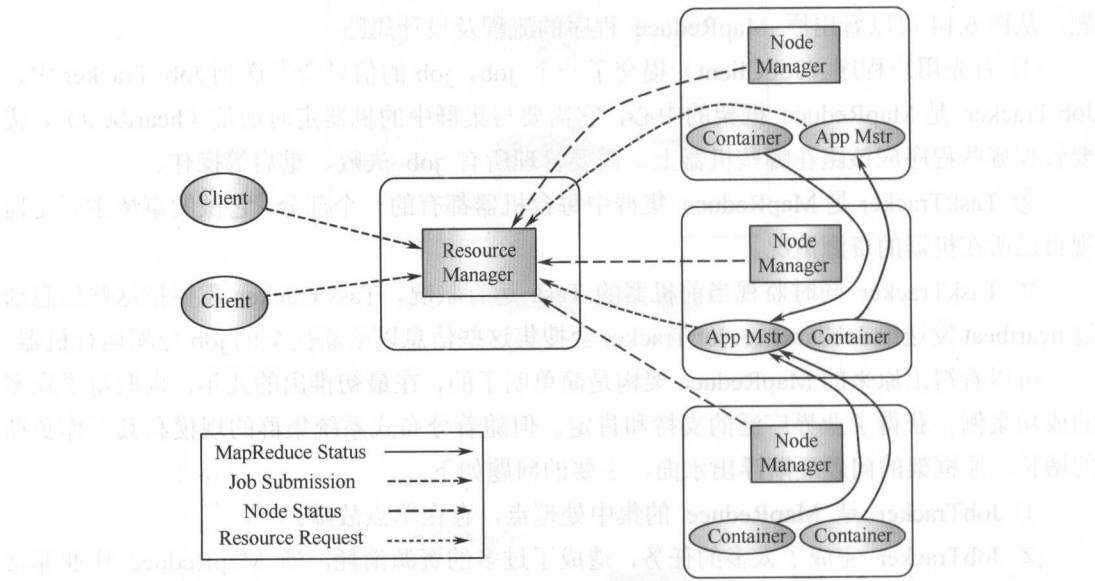


图 6.19 MapReduceV2 架构图

重构 MapReduce 架构的基本思想是将 JobTracker 分离为两个独立的组件：资源管理和任务调度/监控。新的资源管理器 ResourceManager 管理全局计算资源的分配，每一个应用的 ApplicationMaster 负责其调度和协调。一个应用程序可以是传统的 MapReduce 任务中单个 job，也可以是由任务组成的有向无环图（DAG）。ResourceManager 和每台机器上管理用户进程的 NodeManager 服务，一起组成了计算框架。每个应用程序都有一个 ApplicationMaster，它与此框架指定的类库一起，负责与 ResourceManager 协商资源，并协同 NodeManager 执行和监控任务。

ResourceManager 支持多层次应用队列，可以给这些队列指定一定百分比的资源配额。ResourceManager 是一个纯粹的调度器，它不负责应用运行状态的监控和跟踪。并且，它也不负责重启因为软件或硬件原因导致失败的任务。

ResourceManager 的调度功能是基于应用程序的资源请求量的，每个应用都会有多种资源请求类型，表示资源容器（container）请求的资源。请求的资源类型包括内存、CPU、硬盘、网络等。注意，这和以前的基于槽位的 Slot 模型相比有显著的区别，槽位的管理方式给集群的资源利用率带来了很大的负面影响。ResourceManager 提供一个调度策略插件，负

责将集群中的资源划分到不同的队列中。资源调度插件可以基于现有的 CapacityScheduler 和 FairScheduler 等进行设计。

NodeManager 作为集群每个节点上框架的代理，它负责启动应用程序的容器（container），监控资源的使用情况（CPU、内存、磁盘、网络），并汇报给资源调度器。

每个应用的 ApplicationMaster 负责和资源调度器协商资源，启动任务，跟踪任务运行状态并监控进度，处理任务失败的情况。

3. Yarn 与 MRv1

下面将新的 MapReduce 框架与老的框架做对比，新的框架在以下各个方面相对于老的框架带来了性能提升。

（1）扩展性

将资源的管理和应用生命周期的管理分离后，系统具有更好的可扩展性。原来的 Hadoop MapReduce 框架中的 JobTracker 耗费了大量的资源来跟踪应用生命周期，这是造成软件失误的重要原因。将 JobTracker 的角色由一个应用实体（ApplicationMaster）来指定是一个重大的改进。

从当前的硬件发展趋势来看，可扩展性显得尤为重要，目前已经部署 Hadoop MapReduce 的集群达 4000 台。然而，2009 年的 4000 台机器（即 8 核，16GB 内存，4TB 磁盘）的处理能力只有 2011 年的 4000 台机器（16 核，48GB 内存，24TB 磁盘）的一半。此外，运营成本迫使我们去运行一个拥有 6000 台机器的，比以往任何时候都大的集群。

（2）可用性

ResourceManager——资源管理器使用 Apache ZooKeeper 来进行 fail-over。当资源管理器挂掉后，备用的资源管理器可以从 ZooKeeper 中保存的集群状态中快速恢复。fail-over 后的备用资源管理器（通过 ResourceManager 中的一个模块——ApplicationsMasters，注意不是 ApplicationMaster）可以重启所有队列中正在运行的应用。

ApplicationMaster ——MapReduce 2.0 支持应用为 ApplicationMaster 指定检查点的能力。ApplicationMaster 可以从 HDFS 上保存的状态中进行失败恢复。

（3）兼容性

MapReduce 2.0 使用兼容的协议以允许不同版本的服务器和客户端通信。在未来的版本中，将支持对集群的更新进行回滚，这在可操作性上是一次重要提升。

（4）创新和灵活性

架构方面一个重大的提议是 MapReduce 成为一个客户端的库。计算框架（资源管理器和节点管理器）变得很通用，完全不受 MapReduce 编程模型的约束。

这使得终端用户可以在一个集群上同时使用不同版本的 MapReduce。这为各个应用的 bug 修复、改进和添加新功能提供了极大的灵活性，因为不再需要升级整个集群了。它还允许终端用户在自己的计划下升级各自应用的 MapReduce 版本，显著地提高了集群的可操作性。

能运行用户定义的 MapReduce 版本的能力促进了创新，而不影响集群的稳定性。可将 Hadoop 在线原型的特点整合到用户的 MapReduce 版本中，而不影响其他用户。

(5) 集群资源利用率

MapReduce 2.0 框架中，ResourceManager 使用了一种通用的概念来调度和分配各个应用的资源。集群中的每个机器在概念上由内存、CPU、I/O 带宽等资源组成。每台机器是可替代的，根据应用的资源请求类型，可以作为容器（container）分配到各个应用。在同一台机器上，容器作为一组进程的集合，在逻辑上与其他的容器独立，提供了强大的多租户支持能力。因此，它消除了目前的 Hadoop MapReduce 框架中固定 Map 和 Reduce 槽位分配的概念。由于在集群的不同时间点，Map 或者 Reduce 都可能稀缺，因此固定槽位的分配方式会显著降低集群的资源利用率。

(6) 支持除 MapReduce 外的编程范式

MapReduce 2.0 提供了一个完全通用的计算框架以支持 MapReduce 和其他的编程模式。该框架允许用户自定义 ApplicationMaster 来实现用户指定的框架。

因此，它支持多种编程范式，如 MapReduce、MPI、MasterWorker 和迭代模型等运行在同一个集群上，并允许每个应用使用适当的框架。这对某些需要结合 MapReduce 和其他自定义框架的应用是相当重要的，如 K-means、Page-Rank 等。

6.3 Hadoop 相关生态系统

本节简要介绍 Hadoop 生态系统的另外几个重要组成部分（图 6.20），在这里我们只介绍这些组件的功能和作用。

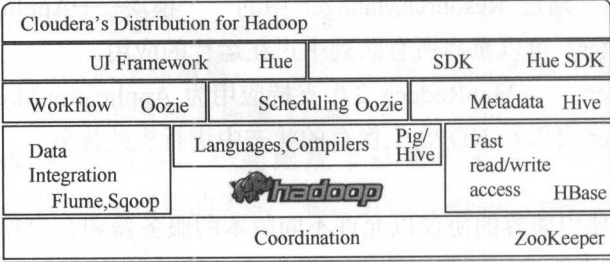


图 6.20 Hadoop 生态系统组成

6.3.1 交互式数据查询分析

Hadoop HDFS 中存储了海量数据，可以想象，如果直接访问这些数据将会给数据的访问人员带来很大的困难，而且数据的安全性也会受到威胁。然而庆幸的是，开源社区专门为 Hadoop 开发了一些交互式数据查询、分析工具。

1. Hive

Hive 是基于 Hadoop 的数据仓库工具，可以将结构化的数据文件映射为一张数据库表，并提供简单的 SQL 查询功能，可以将 SQL 语句转换为 MapReduce 任务运行。其优点是学习成本低，可以通过类 SQL 语句快速实现简单的 MapReduce 统计，不必开发专门的 MapReduce 应用，十分适合数据仓库的统计分析。在上一节介绍的 WordCount 场景中，使用 Hive 的话，一句话就能得到结果。

首先将存储在 HDFS 中的文件导入 Hive 表中，然后执行下面的语句：

```
hive> select count(name) from wordcount_table;
```

2. Impala

在实时性要求不是很高的应用场景中，比如，月度统计报表生成等，基于传统的 Hadoop MapReduce 来处理海量大数据（包括使用 Hive），在各方面表现都还不错，只需要离线处理数据，然后存储结果即可。在一些实时性要求相对较高的应用场景中，处理时间能够在原有的基础上大幅度减少，能很好地提升用户体验。对于大数据的实时性要求，其实是相对的，比如，传统使用 MapReduce 计算框架处理 PB 级别的查询分析请求，可能耗时 30 分钟甚至更多，但是如果能够使这个延迟大大降低，如 3 分钟计算出结果，这是很令人震撼的。Impala 就是基于这样的需求驱动而出现的。

Impala 是 Cloudera 公司主导开发的新型查询系统，它提供 SQL 语义，能查询存储在 Hadoop 的 HDFS 和 HBase 中的 PB 级大数据。已有的 Hive 系统虽然也提供了 SQL 语义，但由于 Hive 底层执行使用的是 MapReduce 引擎，仍然是一个批处理过程，难以满足查询的交互性。相比之下，Impala 的最大优势、最大卖点就是它的快速。

Impala 的架构设计如图 6.21 所示。

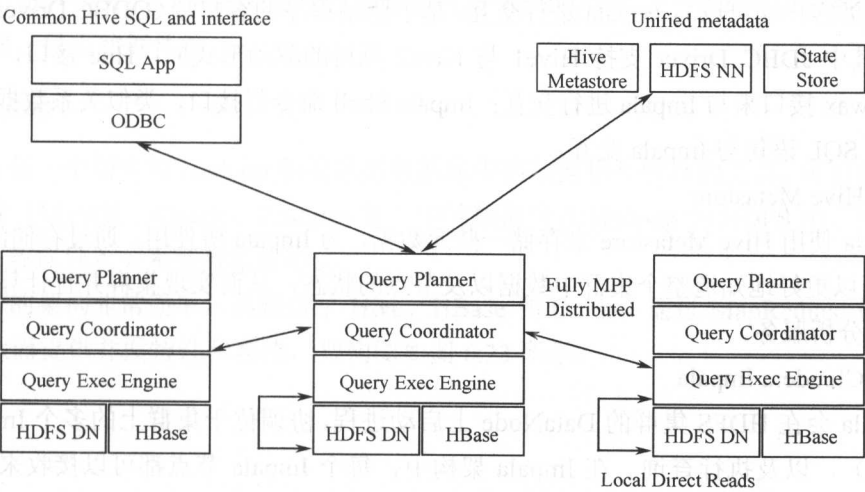


图 6.21 Impala 架构设计图

可以看出，位于 DataNode 上的每个 `impalad` 进程，都具有 Query Planner、Query Coordinator、Query Exec Engine 这几个组件，每个 Impala 节点在功能集合上是对等的，也就是说，任何一个节点都能接收外部查询请求。当有一个节点发生故障后，其他节点仍然能够接管，这还要得益于在 HDFS 上数据的副本是冗余的，只要数据能够取得，某些挂掉的 `impalad` 进程所在节点的数据，在整个 HDFS 中只要还存在副本（`impalad` 进程正常的节点），还是可以提供计算的。除非，当多个 `impalad` 进程挂掉了，恰好此时的查询请求要操作的数据所在的节点，都没有 `impalad` 进程，这肯定是无法计算了。

Cloudera Impala 在实际应用场景中所处的位置如图 6.22 所示。

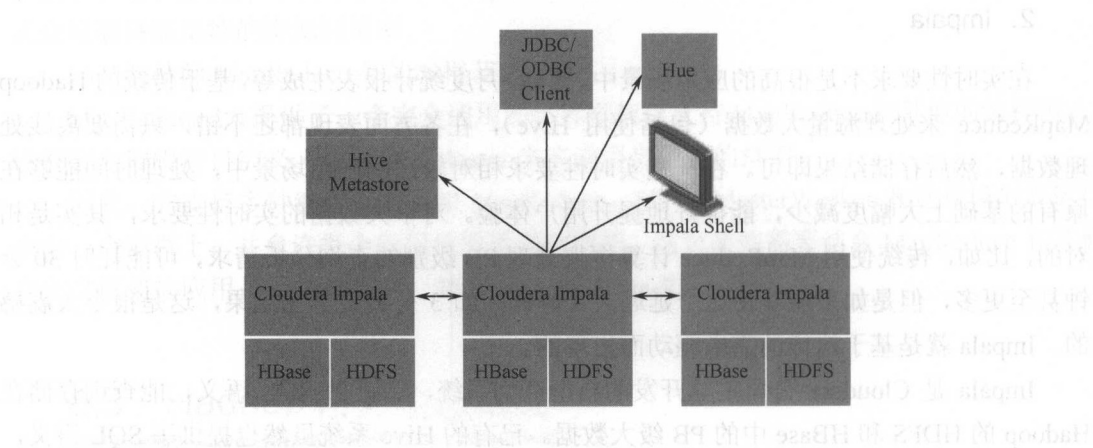


图 6.22 Impala 在实际应用场景中所处的位置示意图

对图 6.22 中所示的各个组件的简要说明如下。

(1) 客户端

有三类客户端可以与 Impala 进行交互：基于驱动程序的客户端（ODBC Driver 和 JDBC Driver，其中 JDBC Driver 支持 Hive1 与 Hive2 风格的驱动形式）；Hue 接口，可以通过 Hue Beeswax 接口来与 Impala 进行交互；Impala Shell 命令行接口，类似关系数据库，可以直接使用 SQL 语句与 Impala 交互。

(2) Hive Metastore

Impala 使用 Hive Metastore 来存储一些元数据，为 Impala 所使用，通过存储的元数据，Impala 可以更好地知道整个集群中数据以及节点的状态，从而实现集群并行计算，对外部提供查询分析服务。

(3) Cloudera Impala

Impala 会在 HDFS 集群的 DataNode 上启动进程，协调位于集群上的多个 Impala 进程（`Impalad`），以及执行查询。在 Impala 架构中，每个 Impala 节点都可以接收来自客户端的查询请求，然后负责解析查询，生成查询计划，并进行优化，协调查询请求在其他的多个 Impala 节点上并行执行，最后由负责接收查询请求的 Impala 节点来汇总结果，响应

客户端。

(4) HBase 和 HDFS

HBase 和 HDFS 存储着实际需要查询的大数据。

Impala 目前在 SQL 解析方面还有优化的余地, 当前的问题, 一个是 SQL 解析速度很慢, 另一个是如果 SQL 比较复杂的话存在硬解析的问题, 非常耗时, 和现在更加成熟的关系数据库 Oracle、MySQL 等相比还有一定差距。

3. Pig

Pig 是一个基于 Hadoop 的大规模数据分析工具, 它提供的 SQL-LIKE 语言叫 Pig Latin, 该语言的编译器会把类 SQL 的数据分析请求转换为一系列经过优化处理的 MapReduce 运算。Twitter 甚至基于 Pig 实现了一个大规模机器学习平台。Pig 是一种编程语言, 它简化了 Hadoop 常见的工作任务。Pig 可加载数据、表达转换数据以及存储最终结果。Pig 内置的操作使得半结构化数据变得有意义(如日志文件)。同时 Pig 可扩展使用 Java 中添加的自定义数据类型并支持数据转换。

6.3.2 数据收集、转换工具

大量数据的收集与转换工作对于 Hadoop 来说也是件轻松的事, 因为它有专门的数据收集、转换工具的支持, 大量数据的采集和存储(如日志文件)往往需要经过一系列的处理(数据 ETL), 有了这些工具的支持就使得工作得以简化。下面介绍两个常用的工具。

1. Flume

Flume 是一个分布、可靠、高可用的海量日志聚合的系统, 可用于日志数据收集、日志数据处理、日志数据传输。

2. Sqoop

Sqoop 是一个用来将 Hadoop 和关系型数据库中的数据相互转移的工具, 可以将一个关系型数据库(MySQL、Oracle、Postgres 等)中的数据导入 Hadoop 的 HDFS 中, 也可以将 HDFS 中的数据导入关系型数据库中。

Sqoop 的架构非常简单, 其整合了 Hive、HBase 和 Oozie, 通过 MapReduce 任务来传输数据, 从而提供并发特性和容错, 架构图如图 6.23 所示。

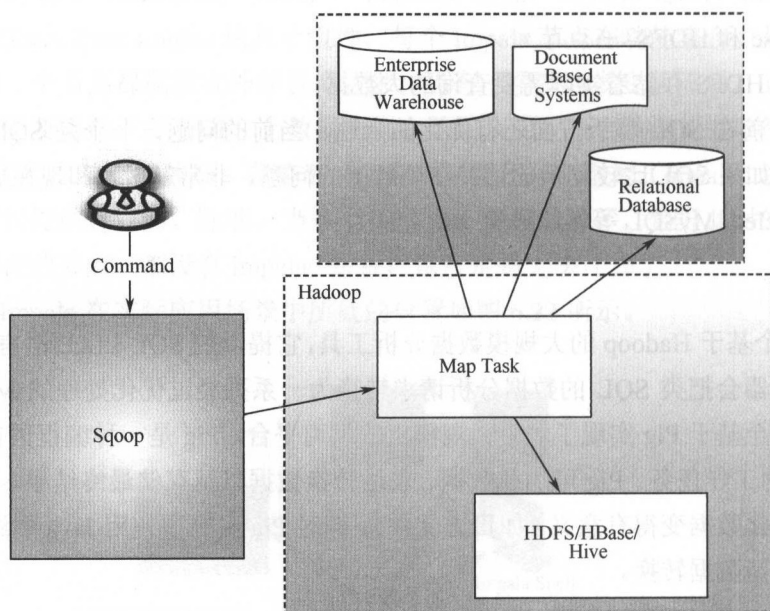


图 6.23 Sqoop 架构图

6.3.3 机器学习工具

Apache Mahout 是基于 Hadoop 的机器学习和数据挖掘的一个分布式框架。Mahout 用 MapReduce 实现了部分数据挖掘算法，解决了并行挖掘的问题。

6.3.4 集群管理与监控

随着 Hadoop 越来越普及，对合适的管理平台的需求成为当前亟待解决的问题。已经有几个商业性的 Hadoop 管理平台，如 Cloudera Enterprise Manager，但 Apache Ambari 是第一个开源实现。

Apache Ambari 是一种基于 Web 的工具，支持 Apache Hadoop 集群的供应、管理和监控。Ambari 目前已支持大多数 Hadoop 组件，包括 HDFS、MapReduce、Hive、Pig、HBase、ZooKeeper、Sqoop 和 HCatalog 等。

Apache Ambari 支持 HDFS、MapReduce、Hive、Pig、HBase、ZooKeeper、Sqoop 和 HCatalog 等的集中管理。Ambari 主要取得了以下成绩。

① 通过一步一步的安装向导简化了集群供应。

② 预先配置好关键的运维指标 (metrics)，可以直接查看 Hadoop Core (HDFS 和 MapReduce) 及相关项目 (如 HBase、Hive 和 HCatalog) 是否健康。

- ③ 支持作业与任务执行的可视化与分析，能够更好地查看依赖和性能。
 - ④ 通过一个完整的 Restful API 把监控信息暴露出来，集成了现有的运维工具。
 - ⑤ 用户界面非常直观，用户可以轻松有效地查看信息并控制集群。
 - ⑥ Ambari 使用 Ganglia 收集度量指标，用 Nagios 支持系统报警，当需要引起管理员的关注时（比如，节点停机或磁盘剩余空间不足等问题），系统将向其发送邮件。
- 此外，Ambari 能够安装安全的（基于 Kerberos）Hadoop 集群，以此实现了对 Hadoop 安全的支持，提供了基于角色的用户认证、授权和审计功能，并为用户管理集成了 LDAP 和 Active Director（图 6.24）。

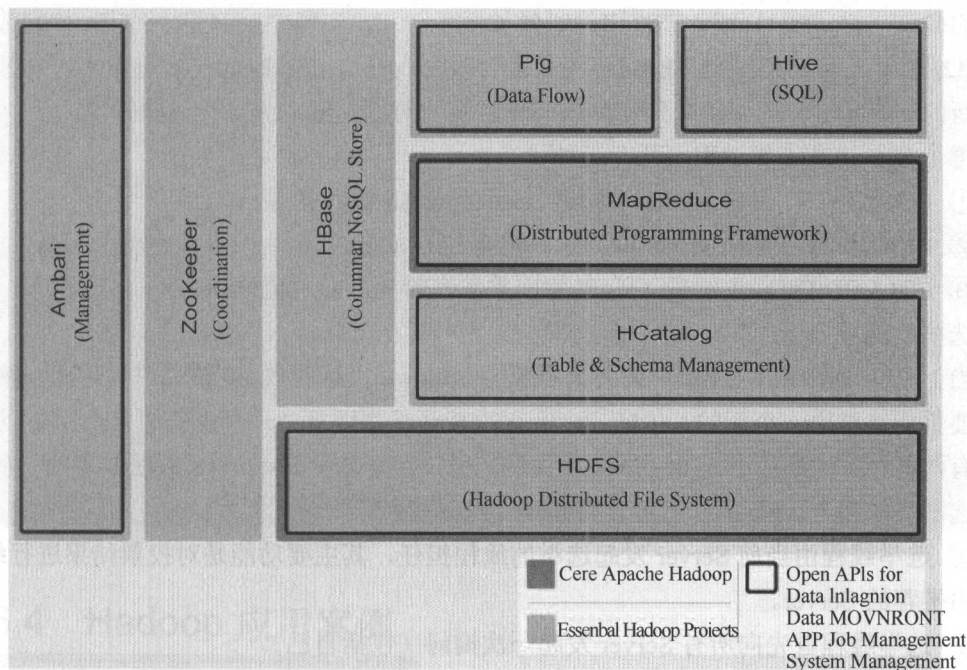


图 6.24 Ambari 架构图

6.3.5 其他工具

Apache ZooKeeper 分布式服务框架是 Apache Hadoop 的一个子项目，它主要用来解决分布式应用中经常遇到的一些数据管理问题，如统一命名服务、状态同步服务、集群管理、分布式应用配置项的管理等。

ZooKeeper 从设计模式角度来看，是一个基于观察者模式设计的分布式服务管理框架，它负责存储和管理大家都关心的数据，然后接受观察者的注册，一旦这些数据的状态发生变化，ZooKeeper 就通知已经在 ZooKeeper 上注册的观察者并做出相应的反应，从而实现集群中类似 Master/Slave 的管理模式。ZooKeeper 作为 Hadoop 项目中的一个子项目，是 Hadoop 集群管理的一个必不可少的模块，它主要用来控制集群中的数据，如它管理

Hadoop 集群中的 NameNode, 还有 HBase 中 Master 选举、Server 之间状态同步等。

下面以 Leader 的选举为例介绍 ZooKeeper 的应用场景。

ZooKeeper 的核心是原子广播, 这个机制保证了各个 Server 之间的同步。实现这个机制的协议叫做 Zab 协议。Zab 协议有两种模式, 分别是恢复模式 (选主) 和广播模式 (同步)。当服务启动或者领导者崩溃后, Zab 就进入了恢复模式, 当领导者被选举出来, 且大多数 Server 完成了和 Leader 的状态同步以后, 恢复模式就结束了。状态同步保证了 Leader 和 Server 具有相同的系统状态。

为了保证事务的顺序一致性, ZooKeeper 采用了递增的事务 ID 号 (zxid) 来标识事务。所有的提议 (proposal) 都在被提出的时候加上了 zxid。实现中 zxid 是一个 64 位的数字, 它高 32 位是 epoch 用来标识 Leader 关系是否改变, 每次一个 Leader 被选出来, 它都会有一个新的 epoch, 标识当前属于那个 Leader 的统治时期。低 32 位用于递增计数。

每个 Server 在工作过程中都有三种状态。

- ① LOOKING: 当前 Server 不知道 Leader 是谁, 正在搜寻。
- ② LEADING: 当前 Server 即为选举出来的 Leader。
- ③ FOLLOWING: Leader 已经选举出来, 当前 Server 与之同步。

选举的流程如下:

当 Leader 崩溃或者 Leader 失去大多数的 Follower, 这时候 Zk 进入恢复模式, 恢复模式需要重新选举出一个新的 Leader, 让所有的 Server 都恢复到一个正确的状态。Zk 的选举算法有两种: 一种是基于 basic paxos 实现的, 另一种是基于 fast paxos 算法实现的。系统默认的选举算法为 fast paxos。下面介绍 basic paxos 流程。

① 选举线程由当前 Server 发起选举的线程担任, 其主要功能是对投票结果进行统计, 并选出推荐的 Server。

② 选举线程首先向所有 Server 发起一次询问 (包括自己)。

③ 选举线程收到回复后, 验证是否是自己发起的询问 (验证 zxid 是否一致), 然后获取对方的 ID (myid), 并存储到当前询问对象列表中, 最后获取对方提议的 Leader 相关信息 (ID, zxid), 并将这些信息存储到当次选举的投票记录表中。

④ 收到所有 Server 回复以后, 就计算出 zxid 最大的那个 Server, 并将这个 Server 相关信息设置成下一次要投票的 Server。

⑤ 线程将当前 zxid 最大的 Server 设置为当前 Server 要推荐的 Leader, 如果此时获胜的 Server 获得 $n/2 + 1$ 的 Server 票数, 设置当前推荐的 Leader 为获胜的 Server, 将根据获胜的 Server 相关信息设置自己的状态, 否则, 继续这个过程, 直到 Leader 被选举出来。

Leader 选举关键代码:

```
void findLeader() throws InterruptedException {
    byte[] leader = null;
    try {
        leader = zk.getData(root + "/leader", true, null);
    }
}
```

```

    } catch (Exception e) {
        logger.error(e);
    }
    if (leader != null) {
        following();
    } else {
        String newLeader = null;
        try {
            byte[] localhost = InetAddress.getLocalHost().getAddress();
            newLeader = zk.create(root + "/leader", localhost,
                ZooDefs.Ids.OPEN_ACL_UNSAFE, CreateMode.EPHEMERAL);
        } catch (Exception e) {
            logger.error(e);
        }
        if (newLeader != null) {
            leading();
        } else {
            mutex.wait();
        }
    }
}

```

6.4 Hadoop 应用案例

随着企业的数据量的迅速增长，存储和处理大规模数据已成为企业的迫切需求。Hadoop 作为开源的云计算平台，已引起了学术界和企业的普遍兴趣。在学术方面，Hadoop 得到了各科研院所的广泛关注，多所著名大学加入 Hadoop 集群的研究中，其中包括斯坦福大学、加州大学伯克利分校、康奈尔大学、卡耐基·梅隆大学、普渡大学等。一些国内高校和科研院所，如中科院计算所、清华大学、中国人民大学等也开始对 Hadoop 展开相关研究，研究内容涉及 Hadoop 的数据存储、资源管理、作业调度、性能优化、系统可用性和安全性等多个方面。在商业方面，Hadoop 技术已经在互联网领域得到了广泛的应用。互联网公司往往需要存储海量的数据并对其进行处理，这正是 Hadoop 的强项。如 Facebook 使用 Hadoop 存储内部的日志副本以及数据挖掘和日志统计，Yahoo 利用 Hadoop 支持广告系统并处理网页搜索，Twitter 则使用 Hadoop 存储微博数据、日志文件和其他中间数据等。在国内，Hadoop 同样也得到了许多公司的青睐，如百度主要将 Hadoop 应用于日志分析和网页数据库的数据挖掘，阿里巴巴则将 Hadoop 用于商业数据的排序和搜索引擎的优化等。

如何快速搭建一个本地的 Hadoop 环境进行实战演练呢？这里给读者介绍一个产品，由 Hortonworks 发布的一个虚拟机环境——Sandbox，它是一种可移植个人 Hadoop 环境，附带大量交互式 Hadoop 教程。Sandbox 包含来自最新 HDP 发行版的大量改进，封装在虚拟环境中。可以在下面的网址找到：<http://zh.hortonworks.com/products/hortonworks-sandbox/>。

6.5 练习题

1. 简要阐述 Hadoop 及其生态系统的各个组件的功能。
2. 简述 HDFS 的设计思想。
3. 在自己的电脑上搭建一个 Hadoop 环境，并完成 WordCount 程序。
4. 一个文本文件，大约有一万行，每行一个词，要求统计出其中最频繁出现的前 10 个词，请给出思想及时间复杂度分析。

参考文献

- [1] 陆嘉恒. Hadoop 实战 2 版. 北京: 机械工业出版社, 2012.
- [2] HBase 技术介绍, <http://www.searchtb.com/2011/01/understanding-hbase.html>
- [3] Dean J, Ghemawat S. MapReduce: simplified data processing on large clusters[J]. Communications of the ACM, 2008, 51(1): 107-113.
- [4] Borthakur D. HDFS architecture guide[J]. HADOOP APACHE PROJECT <http://hadoop.apache.org/common/docs/current/hdfs design. pdf>, 2008.
- [5] Taylor R C. An overview of the Hadoop/MapReduce/HBase framework and its current applications in bioinformatics[J]. BMC bioinformatics, 2010, 11(Suppl 12): S1.
- [6] George L. HBase: the definitive guide[M]. “O'Reilly Media, Inc.”, 2011.
- [7] White T. Hadoop: The definitive guide[M]. “O'Reilly Media, Inc.”, 2012.
- [8] <http://www.ibm.com/developerworks/cn/opensource/os-cn-hadoop-yarn/>
- [9] <http://hive.apache.org/>
- [10] <http://hbase.apache.org/>
- [11] <http://mahout.apache.org/>
- [12] <http://ambari.apache.org/>
- [13] 大数据技术基础, <http://dblab.xmu.edu.cn/post/1089/>

Spark——大数据统一计算平台

7.1 Spark 简介

7.1.1 Spark

Spark 是 UC Berkeley AMP Lab 开源的类 Hadoop MapReduce 的通用并行计算框架，主要用来加快数据分析的运行和读写速度。Spark 是基于 MapReduce 算法实现的分布式计算，它在拥有 Hadoop MapReduce 所有优点的基础上，其任务的中间结果还可以保存在内存中，不用再读写 HDFS，从而实现快速查询，速度比基于磁盘的系统更快。因此，Spark 在处理迭代算法（如机器学习、图挖掘算法）和交互式数据挖掘算法等方面具有更大的优势。Spark 的架构如图 7.1 所示。

Shark 在 Spark 框架的基础上提供和 Hive 一样的 HiveQL 的命令接口。它与 Apache Hive 完全兼容，支持低延时、通过内存计算交互式查询，使用户在不改变查询语句和数据的情况下执行 Hive 查询，并且处理速度比 Hive 快百倍。由于与 Hive 兼容，Shark 可以处理所有支持 Hadoop 存储 API 系统的数据，包括 HDFS、Amazon S3 等。同时，它也支持多种数据格式，例如文本、JSON、XML 和二进制序列文件等。Shark 可以通过 UDF 用户自定义函数实现特定的数据分析学习算法，使得 SQL 数据查询和运算分析功能结合在一起。目前，已经终止了对 Shark 项目的开发，在 Spark 1.0 中提出了 Spark SQL。Spark SQL 包含 Shark 的所有特性，在这基础上，减少了 Hive 的依赖。

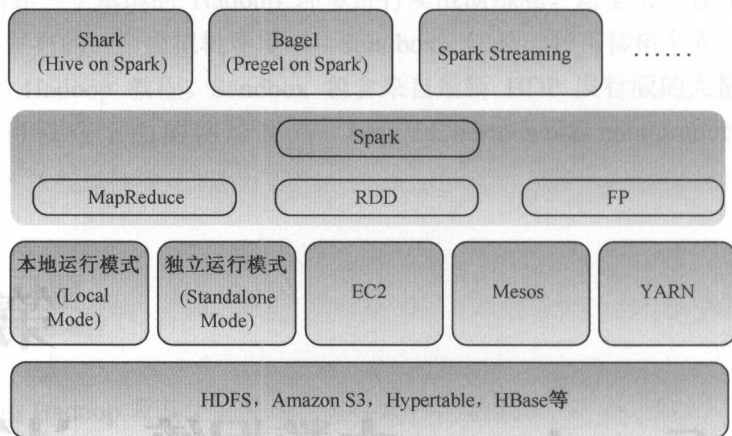


图 7.1 Spark 架构图

与 Google 的 Pregel 相类似，Bagel 是基于 Spark 的轻量级的图计算框架的实现。它可以利用 Spark 来进行图计算，是一个非常有用的小项目。

Spark Streaming 是构建在 Spark 上的实时流处理计算框架。它将流数据以秒为单位分成一段一段的时间片，以类似于批处理的方式去处理每一个时间片的流式数据。该种处理方式可以使 Spark Streaming 在处理数据时兼顾批量处理和实时数据处理的逻辑和算法，因此 Spark Streaming 可以用于一些需要处理历史数据和实时数据联合分析的应用场景。

Spark 的核心概念是 RDD (Resilient Distributed Dataset, 弹性分布式数据集)。Spark 所操作的所有数据集都是包装成 RDD 来进行操作的。RDD 表示已经分区、不可改变的能够被并行操作的数据集合，不同的数据集格式对应不同的 RDD 实现。RDD 必须是可序列化的。与 MapReduce 相比，每次 RDD 数据集的操作结束以后都可以存储至内存，下一个操作可以直接从内存中输入，从而节省了大量的磁盘 I/O 操作。RDD 有两个重要的特征，一是弹性，在计算过程中，当遇到内存不足时，它可以与硬盘进行数据交换，虽然在某种程度上要降低性能，但可以保证计算的顺利进行；二是分布式的，它可以分布在多台机器上进行并行计算。

Spark 使用 Scala 语言实现，为 Scala 和 Java 提供了 API，用户还可使用 Scala 命令行查询大数据集。Spark 引入了基于内存的集群计算，它允许应用在内存中保存工作集以便高效地重复利用，它支持多种数据处理应用，同时它也保存了 MapReduce 的重要特性，如高容错性、数据本地化和大规模处理等。

而 RDD 也表现为一个 Scala 对象，可以由一个文件来创建，它们分布在一个集群内，是不可变的对象切分集；通过并行处理 (Map、filter、join) 固定数据 (BaseRDD) 创建模型，生成 Transformed RDD；在发生故障时可使用 RDD 血统信息自动重建；可以控制存储级别，以便再利用。

Spark 最初设计在 Apache Mesos 和 Yarn 两个资源管理框架上运行。Spark 的本地运行

模式和独立运行模式可方便调试。

Spark 底层的数据存储可以支持多种框架,例如 HDFS、Amazon S3、HBase 等。Spark 将这些数据集封闭成 RDD 进行操作。例如 Spark 可以兼容处理 HDFS 数据文件,其将 HDFS 数据文件包装成可以识别的 RDD 来完成数据抽取和处理。

与 Hadoop 相比,Spark 具有较多的优势。首先,Spark 将处理的中间数据存放至内存中,对于迭代运算效率更高,Spark 里有 RDD 抽象概念,它更适合于迭代运算比较多的 ML 和 DM 运算。其次,Spark 与 Hadoop 相比更通用,Spark 支持两种类型的操作,一种是 Transformation (转换)操作,包括 Map, filter, sample, groupByKey, reduceByKey, union, join, cogroup, mapValues, sort, partitionBy 等多种数据集操作类型,另一种是 Actions (动作)操作,包括 count, collect, Reduce, lookup, save 等多种类型,而 Hadoop 只提供了 Map 和 Reduce 两种操作。再次,Spark 具有更好的容错性。在分布式数据集计算时可以通过两种方式的 checkpoint 来实现容错,一种是 checkpoint data,另一种是 logging the updates,并且采用哪种方式用户可以自己控制。最后,Spark 具有更高的可用性,它提供丰富的 Scala, Java, Python 的 API 和交互式的 Shell。

Spark 还可以与 Hadoop 相结合,支持直接读写 HDFS 数据,同时支持 Spark on Yarn,可以与 MapReduce 在同一集群中运行,共享存储资源和计算。Spark 也可以实现与 Hive 的完全兼容。

Spark 的应用相当广泛,它适用于需要多次操作特定数据集的应用场合,并且次数越多,数据量越大,就越有效。而由于 RDD 的特性,Spark 不适用于异步细粒度更新状态的应用,例如 Web 服务的存储、增量的爬虫和索引等。

7.1.2 BDAS

大数据时代,数据处理的目标主要有以下三个方面。

- ① 通过对历史数据低延时交互式的处理,从而更快地做出决策,例如确定一个站点缓慢的原因并修复。
- ② 通过对流式数据的快速处理,从而可以做出实时决策,例如实时检测并阻止蠕虫。
- ③ 通过对复杂数据的处理,从而做出更好、更准确的决策,例如异常检测、趋势分析等。

而目前开源的数据分析栈更多关注的是保存在磁盘上大型数据集,虽然在批处理上具有较多的优化,但处理速度相对缓慢。AMP 实验室的 BDAS (Berkeley Data Analytics Stack) 的目标就是将批处理、交互式处理和流处理在一个堆栈里实现,能够比较容易地开发复杂的算法,并且与目前开源的生态系统 Hadoop/HDFS 相兼容。

BDAS (伯克利数据分析栈)主要有四部分,分别是资源管理层、数据管理层、数据处理层以及应用层。在资源管理层实现基础设施框架的共享、数据中心的多级程序;数据管理层实现基础框架间有效的数据共享;数据处理层包括内存数据处理以及处理的时间、

质量和成本等；应用层主要包括一些新的应用，如 AMP-Genomics、Carat 等。BDAS 的架构如图 7.2 所示。

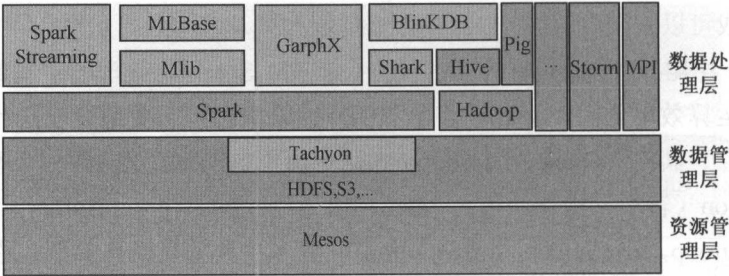


图 7.2 BDAS 架构图

BDAS 软件框架与 Hadoop 软件堆栈是互补的关系，Spark 相当于 MapReduce，Mesos+Tachyon 相当于 HDFS+Yarn，它们可以灵活组合。BDAS 与 Hadoop 和组合既解决了大数据带来的挑战，同时又能较好地完成批量化的处理，把它们应用在同一个系统中，使得数据处理的过程更加方便、快捷。

资源管理层使用 Mesos 来使多框架可以共享同一个集群资源。Mesos 是一个开源集群资源管理和调度系统，允许多个框架共享集群，并与现在的开放分析堆栈相兼容，目前支持的系统有 Spark, Hadoop, TorQue, MPI。与 Mesos 类似的系统有 Hadoop 的 Yarn, Google 的 Borg, 腾讯的 torca 等。

Tachyon 是 AMP 实验室为 BDAS 所开发的一个基于内存的分布式文件系统。Tachyon 同时支持 Spark 和 Hadoop，并提供与 HDFS 兼容的 API 接口，具有高吞吐量和容错的内存存储。在容错性方面，Tachyon 没有使用 Replica 复制内存数据，而是采用和 Spark RDD 类似的 Lineage 用于灾难恢复。

MLBase 在 Spark 生态圈里负责机器学习部分，与其相似的系统有 Weka, Mahout 等，但是 Weka 是一个单机系统，而 MLBase 是分布式的，Weka 和 Mahout 都需要用户具备一些机器学习的相关技能，选择相应的算法和参数进行数据处理，而 MLBase 是自动化的，它提供了不同程度的接口，开发了可扩展的 ML 算法，使得非机器学习的专家也可以使用 ML。

BlinkDB 是应用于大规模数据处理的并行近似查询引擎，它允许用户通过权衡数据精度来提升查询响应时间，其数据的精度被控制在允许的误差范围内。BlinkDB 已经在 Facebook 上进行了测试。BlinkDB 使用两个核心思想：一个是自适应优化框架，从原始数据随着时间的推移建立并维护一组多维样本；另一个是动态样本选择策略，选择一个适当大小的示例基于查询的准确性和（或）响应时间需求。

BDAS 中的 Spark Streaming, Spark SQL, Mlib, Graphx 等，将在后续章节进行介绍。

7.2 RDD

RDD (Resilient Distributed Dataset) 是 Spark 最为核心的概念。近年来, 有关分布式运算的编程框架和模型 (例如 MapReduce、Dryad) 被大量运用于处理不断增长的数据, 这些系统具有高容错、良好的负载均衡等优点, 使得大部分用户都可以使用这些系统进行大数据的处理。但是几乎所有的现有的分布式批处理计算系统都基于非循环式的数据流模型。这意味着每一次计算过程都必然包含着从存储中读取数据到计算完成之后将结果写入存储中的过程, 这样的模型使得那些需要重复使用一个特定的数据集的迭代算法无法高效地运行, 基于 RDD 的 Spark 正是为了解决这一类问题而诞生的。

RDD 的设计理念是在保留 MapReduce 等数据流模型框架的优点的同时 (自动容错、本地优化分配 locality-aware scheduling、可扩展性), 使得用户可以明确地将一部分数据缓存在内存中, 以大大加速对这部分数据之后的查询和计算过程。

RDD 可以被认为是提供了一种高度限制 (只读, 只能从稳定的存储或已有的 RDD 创建) 的共享内存, 但是这些限制可以使得自动容错的开支变得很低。RDD 使用了一种称为“血统”的容错机制, 即每一个 RDD 都包含关于它是如何从其他 RDD 变换过来的以及如何重建某一块数据的信息。

在 RDD 之前, 也有一些模型被创造出来解决数据流模型中存在的缺点, 例如 Google 的 Pregel (迭代图计算框架)、Twister 和 HaLoop (迭代 MapReduce 框架) 等, 但这些系统只能被应用于一些非常有限和特殊的场景中。而 RDD 提供了一种更为通用的迭代并行计算框架, 使得用户能够显式地控制计算的中间结果, 然后将其自由地运用于之后的计算。

7.2.1 RDD 基本概念

RDD 是只读的、分区记录的集合。RDD 只能基于在稳定物理存储中的数据集和其他已有的 RDD 上执行确定性操作来创建。这些确定性操作称为转换 (Transformation), 如 Map、filter、groupBy、join (转换不是程序开发人员在 RDD 上执行的操作)。

RDD 不需要物化。RDD 含有如何从其他 RDD 衍生 (即计算) 出本 RDD 的相关信息 (即 Lineage), 据此可以从物理存储的数据计算出相应的 RDD 分区。同时, RDD 还提供了一组丰富的操作来操作这些数据。在这些操作中, 诸如 Map、flatMap、filter 等转换操作实现了 monad 模式, 很好地契合了 Scala 的集合操作。除此之外, RDD 还提供了诸如 join、groupBy、reduceByKey 等更为方便的操作 (注意, reduceByKey 是 Action, 而非 Transformation), 以支持常见的数据运算。表 7.1 给出了 RDD 上的常见操作, Seq[T] 表示类型为 T 的一系列元素。

表 7.1 在 Spark 上实现的 RDD 的操作

Transformation (转换)	<div>Map($f : T \rightarrow U$) : $RDD[T] \rightarrow RDD[U]$</div> <div>filter($f : T \rightarrow \text{Bool}$) : $RDD[T] \rightarrow RDD[T]$</div> <div>flatMap($f : T \rightarrow \text{Seq}[U]$) : $RDD[T] \rightarrow RDD[U]$</div> <div>sample($\text{fraction} : \text{Float}$) : $RDD[T] \rightarrow RDD[T]$ (Deterministic sampling)</div> <div>groupByKey() : $RDD[(K, V)] \rightarrow RDD[(K, \text{Seq}[V])]$</div> <div>reduceByKey($f : (V, V) \rightarrow V$) : $RDD[(K, V)] \rightarrow RDD[(K, V)]$</div> <div>union() : ($RDD[T]; RDD[T]$) $\rightarrow RDD[T]$</div> <div>join() : ($RDD[(K, V)]; RDD[(K, W)]$) $\rightarrow RDD[(K, (V, W))]$</div> <div>cogroup() : ($RDD[(K, V)]; RDD[(K, W)]$) $\rightarrow RDD[(K, (\text{Seq}[V], \text{Seq}[W]))]$</div> <div>crossProduct() : ($RDD[T]; RDD[U]$) $\rightarrow RDD[(T, U)]$</div> <div>mapValues($f : V \rightarrow W$) : $RDD[(K, V)] \rightarrow RDD[(K, W)]$ (Preserves partitioning)</div> <div>sort($c : \text{Comparator}[K]$) : $RDD[(K, V)] \rightarrow RDD[(K, V)]$</div> <div>partitionBy($p : \text{Partitioner}[K]$) : $RDD[(K, V)] \rightarrow RDD[(K, V)]$</div>
Action (动作)	<div>count() : $RDD[T] \rightarrow \text{Long}$</div> <div>collect() : $RDD[T] \rightarrow \text{Seq}[T]$</div> <div>Reduce($f : (T;T) \rightarrow T$) : $RDD[T] \rightarrow T$</div> <div>lookup($k : K$) : $RDD[(K, V)] \rightarrow \text{Seq}[V]$ (On hash/range partitioned RDDs)</div> <div>save($\text{path} : \text{String}$) : Outputs RDD to a storage system, e.g., HDFS</div>

通常来讲，针对数据处理有几种常见模型，包括 Iterative Algorithms, Relational Queries, MapReduce, Stream Processing。例如 Hadoop MapReduce 采用了 MapReduce 模型，Storm 则采用了 Stream Processing 模型。RDD 混合了这四种模型，使得 Spark 可以应用于各种大数据处理场景。

一个 RDD 可以包含多个分区，每个分区就是一个 dataset 片段。RDD 可以相互依赖。如果 RDD 的每个分区最多只能被一个 Child RDD 的一个分区使用，则称为 narrow dependency；若多个 Child RDD 分区都可以依赖，则称为 wide dependency。不同的操作依据其特性，可能会产生不同的依赖。例如 Map 操作会产生 narrow dependency，而 join 操作则产生 wide dependency。

Spark 之所以将依赖分为 narrow 与 wide，基于以下两点原因。

首先，narrow dependency 可以支持在同一个 cluster node 上以管道形式执行多条命令，例如在执行了 Map 后，紧接着执行 filter。相反，wide dependency 需要所有的父分区都是可用的，可能还需要调用类似 MapReduce 的操作进行跨节点传递。

其次，从失败恢复的角度考虑，narrow dependency 的失败恢复更有效，因为它只需要重新计算丢失的 Parent Partition 即可，而且可以并行地在不同节点进行重计算。而 wide dependency 牵涉 RDD 各级的多个 Parent Partition。图 7.3 说明了 narrow dependency 与 wide dependency 事例之间的区别。

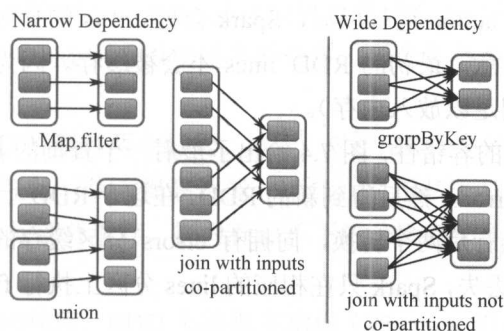


图 7.3 narrow dependency 与 wide dependency 示例

图 7.3 中，一个实线框代表一个 RDD，矩形代表 RDD 的一个分区。

7.2.2 RDD 示例

下面我们通过一个具体示例来阐述 RDD。假定有一个大型网站出错，操作员想要检查 Hadoop 文件系统（HDFS）中的日志文件（TB 级大小）来找出原因。通过使用 Spark，操作员只需要将日志中的错误信息装载到一组节点的内存中，然后执行交互式查询。首先，需要在 Spark 解释器中输入如下 Scala 命令：

```
lines = spark.textFile("hdfs://...")
errors = lines.filter(_.startsWith("ERROR"))
errors.cache()
```

第 1 行从 HDFS 文件定义了一个 RDD（即一个文本行集合），第 2 行获得一个过滤后的 RDD，第 3 行请求将 errors 缓存起来。注意在 Scala 语法中 filter 的参数是一个闭包。

这时集群还没有开始执行任何任务。但是，用户已经可以在这个 RDD 上执行对应的动作了，例如统计错误消息的数目：

```
errors.count()
```

用户还可以在 RDD 上执行更多的转换操作，并使用转换结果，如：

```
// Count errors mentioning MySQL:
errors.filter(_.contains("MySQL")).count()

// Return the time fields of errors mentioning
// HDFS as an array (assuming time is field number 3 in a tab-separated format):
errors.filter(_.contains("HDFS"))
    .map(_.split("\t")(3))
    .collect()
```


使用 errors 的第一个 action 运行以后，Spark 会把 errors 的分区缓存在内存中，极大地加快了后续计算速度。注意，最初的 RDD lines 不会被缓存。因为错误信息可能只占原数据集的很小一部分（小到足以放入内存）。

最后，为了说明模型的容错性，图 7.4 给出了最后一个查询的 Lineage 图。在 RDDlines 上执行 filter 操作，得到 errors，然后得到新的 RDD，在这个 RDD 上执行 collect 操作。Spark 调度器以流水线的方式执行后两个转换，向拥有 errors 分区缓存的节点发送一组任务。此外，如果某个 errors 分区丢失，Spark 只在相应的 lines 分区上执行 filter 操作来重建该 errors 分区。

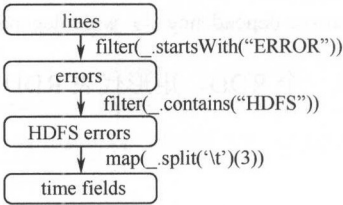


图 7.4 示例中最后查询的 Lineage 图（方框表示 RDD，箭头表示转换）

7.2.3 RDD 与分布式共享内存

为了进一步理解 RDD 是一种分布式的内存抽象，表 7.2 列出了 RDD 与分布式共享内存（Distributed Shared Memory，DSM）的对比。在 DSM 系统中，应用可以向全局地址空间的任意位置进行读写操作。注意这里的 DSM，不仅指传统的共享内存系统，还包括那些通过分布式哈希表或分布式文件系统进行数据共享的系统，比如 Piccolo。DSM 是一种通用的抽象，但这种通用性同时也使得在商用集群上实现有效的容错性更加困难。

RDD 与 DSM 的主要区别在于，不仅可以通过批量转换创建（即写）RDD，还可以对任意内存位置读写。也就是说，RDD 限制应用执行批量写操作，这样有利于实现有效的容错。特别地，RDD 没有检查点开销，因为可以使用 Lineage 来恢复 RDD。而且，失效时只需要重新计算丢失的那些 RDD 分区，可以在不同节点上并行执行，而不需要回滚整个程序。

表 7.2 RDD 与 DSM 对比

对比项目	RDD	DSM
读	批量或细粒度操作	细粒度操作
写	批量转换操作	细粒度操作
一致性	不重要（RDD 是不可改变的）	取决于应用或运行时
容错性	细粒度，低开销（使用 Lineage）	需要检查点操作和程序回滚
落后任务的处理	任务备份	很难处理
任务安排	基于数据存放的位置自动实现	取决于应用程序
当内存不足时	与已有的数据流系统类似	性能较差

注意，通过备份任务的副本，RDD 还可以处理落后任务（即运行很慢的节点），这点与 MapReduce 类似。而 DSM 则难以实现备份任务，因为任务及其副本都需要读写同一个内存位置。

与 DSM 相比，RDD 模型有两个好处。第一，对于 RDD 中的批量操作，运行时将根据数据存放的位置来调度任务，从而提高性能。第二，对于基于扫描的操作，如果内存不足以缓存整个 RDD，就进行部分缓存。把内存放不下的分区存储到磁盘上，此时性能与现有的数据流系统差不多。

最后看一下读操作的粒度。RDD 上的很多动作（如 count 和 collect）都是批量读操作，即扫描整个数据集，可以将任务分配到距离数据最近的节点上。同时，RDD 也支持细粒度操作，即在哈希或范围分区的 RDD 上执行关键字查找。

7.3 Spark SQL

Spark SQL 是支持在 Spark 中使用 SQL、HiveSQL、Scaca 的关系型查询表达式。它的核心组件是一个新增的 RDD 类型 SchemaRDD，它用一个 Schema 来描述行里的所有列的数据类型，它就像是关系型数据库里的一张表。它可以从原有的 RDD 创建，也可以是 Parquet 文件，最重要的是它支持用 HiveQL 从 Hive 里面读取数据。

Spark SQL 对 SQL 语句的处理与关系型数据库对 SQL 语句的处理方法类似，在 Spark SQL 中两个重要的概念是 Tree 和 Rule。处理的过程为：先将 SQL 语句进行解析，形成一个 Tree，后面的绑定、优化等过程都是对语法树进行的操作，操作的方法是采用 Rule，通过模式匹配，对不同类型的节点采用不同的操作。在整个 SQL 语句的处理过程中，语法树和 Rule 相互配合，完成了解析、绑定（在 Spark SQL 中称为 Analysis）、优化、物理计划等过程，最终生成可以执行的物理计划。

在 SparkSql 执行过程中，逻辑计划、物理算子等都可以使用 Tree 来表示。使用 TreeNode 来实现 Tree 的具体操作过程。Spark 定义了 catalyst.trees 日志，通过该日志能够形象地表示出树的结构。TreeNode 可以使用 Scala 的集合操作方法（如 foreach, Map, flatMap, collect 等）进行操作。

TreeNode 可以分成三种类型：UnaryNode（一元节点），其只有一个子节点，例如 Limit、Filter 等操作；BinaryNode（二元节点），其有左右子节点的二叉节点，如 Join、Union 等操作；LeafNode（叶子节点），其没有子节点，主要是用户命令类操作，如 SetCommand。

而在 SparkSQL 的分析器、优化器和 SparkPlan 等各个组件中都用到 Rule。Rule 是一个抽象类，具体的 Rule 实现是通过 RuleExecutor 完成的。Rule 通过定义 batch 和 batches，可以简便、模块化地对 Tree 进行转换操作，Rule 通过定义 Once 和 FixedPoint，可以对 Tree 进行一次操作或多次操作。

如图 7.5 所示，在由解析器（SQLParse）生成的 LogicPlan Tree 时，在 Analyzer

中就定义了多种 Rule 应用到 LogicPlan Tree 上。

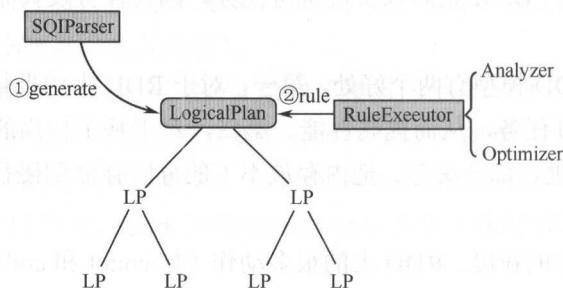


图 7.5 Spark SQL 解析过程

Spark SQL 的核心是把已有的 RDD 带上 Schema 信息, 然后注册成类似 SQL 里的 Table, 对其进行 SQL 查询。这里面主要分两部分, 一是生成 SchemaRDD, 二是执行查询。

对于 Spark SQL 来说, 数据方面, RDD 可以来自任何已有的 RDD, 也可以来自支持的第三方格式, 如 json file、parquet file。SQLContext 会把带 case class 的 RDD 隐式转化为 SchemaRDD。

ExsitingRDD 单例里会反射出 case class 的 attributes, 并把 RDD 的数据转化成 Catalyst 的 GenericRow (Row 和 GenericRow 是 Catalyst 里的行表示模型), 最后返回 RDD[Row], 即一个 SchemaRDD。这里的具体转化逻辑可以参考 ExsitingRDD 的 productToRowRDD 和 convertToCatalyst 方法。

之后可以进行 SchemaRDD 提供的注册 Table 操作、针对 Schema 复写的部分 RDD 转化操作、DSL 操作、saveAs 操作等。

SQLcontext 执行查询的流程如下:

- ① SQL 语句经过 SqlParse 解析成 Unresolved LogicalPlan。
- ② 使用 analyzer 结合数据字典 (catalog) 进行绑定, 生成 resolved LogicalPlan。
- ③ 使用 optimizer 对 resolved LogicalPlan 进行优化, 生成 optimized LogicalPlan。
- ④ 使用 SparkPlan 将 LogicalPlan 转换成 PhysicalPlan。
- ⑤ 使用 prepareForExecution() 将 PhysicalPlan 转换成可执行物理计划。
- ⑥ 使用 execute() 执行可执行物理计划。
- ⑦ 生成 SchemaRDD, SQLContext 里对 SQL 的一个解析和执行流程。

Spark SQL 1.1 总体上由四个模块组成: core、catalyst、Hive、Hive-ThriftServer。

① core 处理数据的输入输出, 从不同的数据源获取数据 (RDD、Parquet、json 等), 将查询结果输出成 SchemaRDD。

② catalyst 处理查询语句的整个处理过程, 包括解析、绑定、优化、物理计划等, 与其说是优化器, 还不如说是查询引擎。

③ Hive 对 Hive 数据的处理。

④ Hive-ThriftServer 提供 CLI 和 JDBC/ODBC 接口。

在这四个模块中，catalyst 处于最核心的部分，其性能优劣将影响整体的性能。由于发展时间尚短，还有很多不足的地方，但其插件式的设计，为未来的发展留下了很大的空间。图 7.6 是 catalyst 的一个设计图。

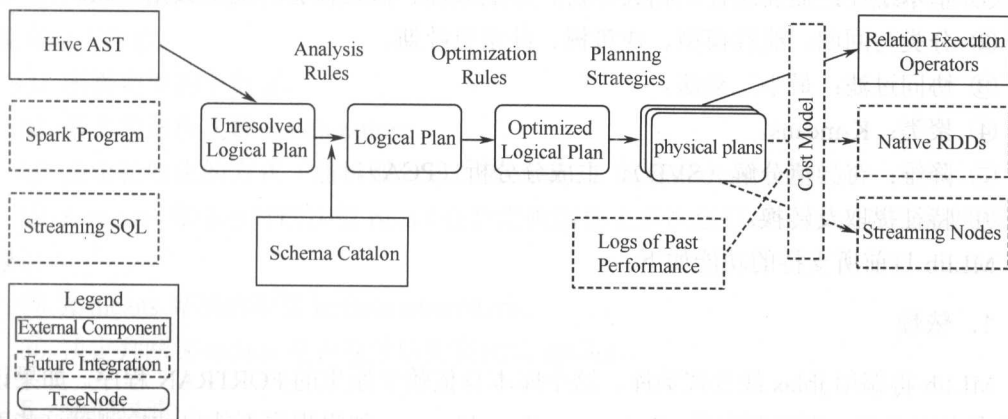


图 7.6 catalyst 设计架构

图 7.6 中虚线部分是以后版本要实现的功能，实线部分是已经实现的功能。catalyst 主要的实现组件如下。

- ① SQLParse，完成 SQL 语句的语法解析功能，目前只提供了一个简单的 SQL 解析器。
- ② Analyzer，主要完成绑定工作，将不同来源的 Unresolved LogicalPlan 和数据元数据（如 Hive metastore、Schema catalog）进行绑定，生成 resolved LogicalPlan。
- ③ Optimizer 对 resolved LogicalPlan 进行优化，生成 optimized LogicalPlan。
- ④ Planner 将 LogicalPlan 转换成 PhysicalPlan。
- ⑤ CostModel 主要根据过去的性能统计数据，选择最佳的物理执行计划。

这些组件的基本实现方法如下。

- ① 先将 SQL 语句通过解析生成 Tree，然后在不同阶段使用不同的 Rule 应用到 Tree 上，通过转换完成各个组件的功能。
- ② Analyzer 使用 Analysis Rules，配合元数据（如 Hive metastore、Schema catalog），完善 Unresolved LogicalPlan 的属性而转换成 resolved LogicalPlan。
- ③ Optimizer 使用 Optimization Rules，对 resolved LogicalPlan 进行合并、列裁剪、过滤器下推等优化作业而转换成 optimized LogicalPlan。

7.4 MLlib

MLlib 是 Spark 对常用的机器学习算法的实现库，也包括相应的测试和数据生成器。

MLlib 目前支持四种常见的机器学习问题：二元分类、回归、聚类以及协同过滤，同时也包括一个底层的梯度下降优化基础算法。

MLlib 目前支持的算法如下。

- ① 基本统计：概要统计、相关分析、分层取样、假设检验、随机数据生成。
- ② 分类与回归：线性模型、决策树、朴素贝叶斯。
- ③ 协同过滤：最小二乘法。
- ④ 聚类：K-means。
- ⑤ 降维：奇异值分解（SVD）、主成分分析（PCA）。
- ⑥ 特征提取及转换。

MLlib 目前所支持的功能如下。

1. 依赖

MLlib 将调用 jblas 线性代数库。这个库本身依赖于原生的 FORTRAN 程序。如果该库在节点中不存在，则需要安装 gfortran runtime library。如果程序不能自动检测到这些库，MLlib 将抛出链接错误的异常。

2. 二元分类

MLlib 目前支持标准模型家族，线性支持向量机和逻辑回归，同时也包括适用于这两个模型家族的 L1 和 L2 正则化变体。二元分类是监督学习的一种。二元分类希望可以将因变量转化成 0 和 1 两种情况，例如生存或者死亡、男性或者女性、有或无、是或否等。这个问题涉及在一组被打过标签的样例上运行一个学习算法，例如一组由（数字）特征和（相关的）类别标签所代表的实体。这个算法将会返回一个训练好的模型，该模型能够对标签未知的新个体进行潜在标签预测。目前 MLlib 可用的二元分类算法有 SVMWithSGD 和 LogisticRegressionWithSGD 两种。

3. 线性回归

线性回归是利用数理统计中的回归分析，来确定两种或两种以上变量间相互依赖的定量关系的一种统计分析方法。它也是一种经典的监督学习问题。每个个体都有一个与之相关联的实数标签（而在二元分类中个体的标签都是二元的），并且我们希望在给出用于表示这些实体的数值特征后，所预测出的标签值可以尽可能接近实际值。MLlib 支持线性回归和与之相关的 L1（lasso）和 L2（ridge）正则化的变体。MLlib 中的回归算法也利用了底层的梯度下降基础算法，输入参数与上述二元分类算法一致。目前可用的线性回归算法有 LinearRegressionWithSGD、RidgeRegressionWithSGD 和 LassoWithSGD 三种。

4. 聚类

聚类算法是一个非监督学习问题，将物理或抽象对象的集合分成由类似的对象组成的多个类的过程称为聚类。由聚类所生成的簇是一组数据对象的集合，这些对象与同一个簇

中的对象彼此相似，与其他簇中的对象相异。MLlib 目前支持的最广泛使用的算法是 K-means 聚类算法。根据事先定义类簇个数，这个算法能对数据进行聚类。K-means 算法是硬聚类算法，是典型的基于原型的目标函数聚类方法的代表，它是数据点到原型的某种距离作为优化的目标函数，利用函数求极值的方法得到迭代运算的调整规则。MLlib 的实现有如下参数。

- ① 所需类簇的个数 K 。
- ② 最大的迭代次数 `maxIterations`。
- ③ 决定初始化的方式（随机或者通过 K-means）的 `initializationMode`。
- ④ K-means 算法运行的次数 `runs`（在给定数据集上多次运行 K-means 算法会返回最优结果）。
- ⑤ K-means 算法的步骤 `initializationSteps`。
- ⑥ 决定判断 K-means 是否收敛的距离阈值 `epsilon`。

5. 协同过滤

协同过滤是推荐系统的一种主要算法。协同过滤主要作用是分析用户兴趣，在用户群中找到指定用户的相似用户，综合相似用户对某一信息的评价，形成系统对指定用户对该信息的喜好程度预测。MLlib 当前支持基于模型的协同过滤，其中用户和商品通过一小组隐性语义因子进行表达，并且这些因子也用于预测缺失的元素。为此，我们用交替最小二乘法（ALS）来学习这些隐性语义因子。在 MLlib 中的实现有如下参数。

- ① 用于并行化计算的分块个数 `numBlocks`。
- ② 模型中隐性语义因子的个数 `rank`。
- ③ 迭代的次数 `iterations`。
- ④ ALS 的正则化参数 `lambda`。
- ⑤ 决定适用数据集的版本（显性反馈 ALS 或者隐性反馈）的参数 `implicitPrefs`。
- ⑥ 决定偏好行为强度的基准参数 `alpha`，该参数是一个针对隐性反馈 ALS 版本的参数。目前 MLlib 可用的协调过滤算法是 ALS。

6. 梯度下降基础算法

梯度下降法是一个最优化算法，非常适用于大型分布式计算。梯度下降算法通过向当前参数值的负梯度方向移动，迭代地找到这个函数的本地最优解。MLlib 以梯度下降作为一个底层的基础算法，在上面开发了各种机器学习算法。梯度下降算法有如下参数。

- ① 用来计算要被优化的函数的随机梯度的类 `gradient`，MLlib 包含常见损失函数的梯度类，包括 `hinge`，`logistic`，`least-squares`，梯度类将训练样本、标签和当前的参数值作为输入。
- ② 每次迭代中更新权重的类 `underater`，MLlib 包含适用于无正则项、L1 正则项和 L2 正则项 3 种情况下的类。
- ③ 表示梯度下降初始步长的数据 `stepSize`，MLlib 中所有的更新器第 t 步的步长等于 $\text{stepSize} / \sqrt{t}$ 。

- ④ 迭代次数 `numIterations`。
- ⑤ 使用 L1 和 L2 正则项时的正则化参数 `regParam`。
- ⑥ 每一次迭代中用来计算梯度的数据百分比 `minBatchFraction`。

目前 MLlib 可用的梯度下降算法只有 `GardientDescent` 算法。在语言方面,用 Scala、Java 和 Python 都可以调用 MLlib 库。

7.5 GraphX

GraphX 是一些常用的图的算法在 Spark 上的实现,同时提供了全面的 API 接口。它可以快速地完成基于度分布的中枢节点发现、基于最大连通图的社区发现、基于三角计数的关系衡量和基于随机游走的用户属性传播等。

得益于 Spark 特有的 RDD, GraphX 可以无缝地与 Spark SQL、MLlib 等结合使用,例如我们可以使用 Spark SQL 进行数据的 ETL 之后交给 GraphX 进行处理,而 GraphX 在计算的时候又可以和 MLlib 结合使用来共同完成深度数据挖掘等人工智能化的操作,这些特性都是其他图计算平台所无法比拟的。

在淘宝, Spark GraphX 不仅广泛应用于用户网络的社区发现、用户影响力、能量传播、标签传播等,而且越来越多地应用到推荐领域的标签推理、人群划分、年龄段预测、商品交易时序跳转等。

从技术层面讲 Spark GraphX 非常适合于微信、微博、社交网络、电子商务、地图导航等产品,所以可以期待 Spark GraphX 在 Facebook、Twitter、Linkedin、腾讯、百度等的大规模应用。

7.6 Spark Streaming

Spark Streaming 是建立在 Spark 上的应用框架。属于 Spark 的核心 API,它支持高吞吐量、支持容错的实时流数据处理。利用 Spark 的底层框架作为其执行基础,并在其上构建了 DStream 的行为抽象。它可以接受来自 Kafka, Flume, Twitter, ZeroMQ 和 TCP Socket 的数据源,利用 DStream 所提供的 API,用户可以在数据流上实时进行 count, join, aggregate 等操作,还可以直接使用内置的机器学习算法、图算法包来处理数据。作为构建于 Spark 之上的应用框架,其继承了 Spark 的编程风格。

Spark Streaming 能够运行在 100 个以上的节点上,同时达到秒级的延迟,具有容错性和高效性,集成了 Spark 的批处理和交互式查询,为实现复杂的算法提供了简单接口。

7.6.1 基本概念

基于 Spark on Yarn 的 Spark Streaming 架构如图 7.7 所示。

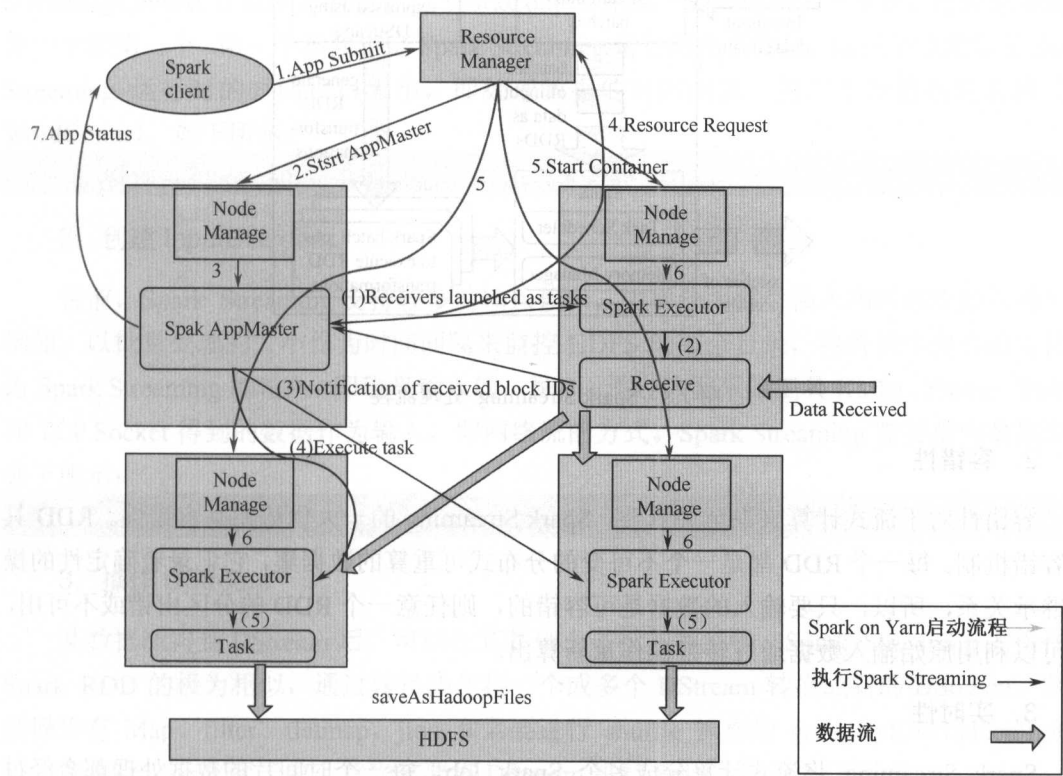


图 7.7 Spark Streaming 架构

1. 计算流程

Spark Streaming 利用批处理引擎 Spark，将流式计算以时间片为单位切分成一系列短小的批处理作业。其将输入的数据流以时间片（秒级）为单位进行拆分，然后以类似批处理的方式处理每个时间片的数据，每一块数据都转换成 Spark 的 RDD，然后使用 RDD 来操作处理每一块数据，每一块都会生成一个 Spark Job 处理，最终结果也返回多块。

Spark Streaming 的处理流程如图 7.8 所示。

Spark Streaming 中将对 DStream 的操作转换为 DStream Graph，对于每个时间片，DStream Graph 都会产生一个 RDD Graph，对于每个输出操作（如 print、foreach 等），Spark Streaming 都会创建一个 Spark Action；对于每个 Spark Action，Spark Streaming 都会产生一个相应的 Spark Job，并交给任务管理器。任务管理器中维护着一个任务队列，Spark Job 存储在这个队列中，任务管理器把 Spark Job 提交给 Spark Scheduler，Spark Scheduler 负责调

度任务到相应的 Spark Executor 上执行。整个流式计算根据业务的需求可以对中间的结果进行叠加，或者存储到外部设备。

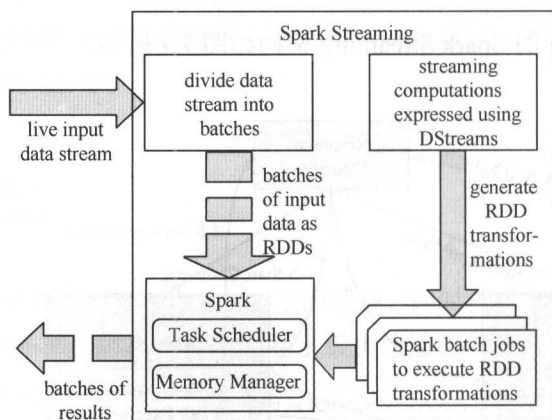


图 7.8 Spark Streaming 处理流程

2. 容错性

容错性对于流式计算来说至关重要。Spark Streaming 的一大优势便是容错性。RDD 具有容错机制。每一个 RDD 都是一个不可变的分布式可重算的数据集，它记录着确定性的操作继承关系，所以，只要输入的数据是可容错的，则任意一个 RDD 的分区出错或不可用，都可以利用原始输入数据通过转换操作重新算出。

3. 实时性

Spark Streaming 将流式计算分成多个 Spark Job，每一个时间片的数据处理都会经过 Spark DAG 图分解和 Spark 任务队列的调度过程。目前，Spark Streaming 最小的时间片选取为 0.5~2 秒，而 Storm 目前的最小延迟是 100ms 左右，因此，Spark Streaming 能够满足所有准实时的计算场景，但对一些实时性要求非常高的场景还不能满足。

4. 可扩展性

Spark 目前在 EC2 上已能够线性扩展到 100 个节点（每个节点 4Core），可以以数秒的延迟处理 6GB/s 的数据量（60M records/s），其吞吐量也比流行的 Storm 高 2~5 倍。

Spark Streaming 通过丰富的 API 和基于内存的调整计算引擎让用户可以把流处理、批处理和交互查询结合起来，它适用于将历史数据和实时数据结合起来分析的应用场合。

7.6.2 编程模型

Spark Streaming 的编程与 Spark 的编程非常类似。对于 Spark，编程是对 RDD 的操作，

而对于 Spark Streaming, 是对 DStream 的操作。下面举例说明对 DStream 的操作。

1. Spark Streaming 初始化

在进行 DStream 操作之前, 需要对 Spark Streaming 进行初始化。同 Spark 初始需要创建 SparkContext 对象一样, 使用 Spark Streaming 就需要创建 StreamingContext 对象。创建 StreamingContext 对象所需的参数与 SparkContext 基本一致, 包括三个参数, 比较重要的是第一个和第三个, 第一个参数指定 Spark Streaming 运行的集群地址, 第三个参数定义 Spark Streaming 运行时的时间窗口大小, 即处理数据的时间间隔。第二个参数指定名称 (如 WordCount)。如下所示:

```
val ssc=new StreamingContext(sparkconf, "WordCount",Seconds(1));
```

2. 创建 InputDStream

目前, Spark Streaming 支持较多的输入接口, 主要分为磁盘输入和网络流输入两种。例如, 以批量处理的大小作为时间间隔来监控 HDFS 的某个目录, 将目录中内容的变化作为 Spark Streaming 的输入, 即磁盘输入的一种, 而通过数据采集工具 Kafka、Flume、Twitter 和 TCP Socket 得到的数据作为输入, 即网络流的方式。Spark Streaming 需要指明数据源, 如下所示:

```
val lines=ssc.socketTextStream(serverIP,serverPort);
```

3. 操作 DStream

从数据源得到 DStream 后, 可以在其基础上进行各种操作。Spark Streaming 的操作与 Spark RDD 的极为相似, 通过转换操作将一个或多个 DStream 转换成新的 DStream。常用的操作有 Map、filter、flatMap, join 和需要进行 shuffle 操作的 groupByKey/reduceByKey 等。在 WordCount 例子中, 首先需要将 DStream 进行 Split 操作, 如下所示:

```
val words=lines.flatMap(_.split(""));
```

然后对相同的单词进行数量统计, 最终得到的即为每一个批处理的中间结果。如下所示:

```
val pairs = words.map(word => (word, 1));
val wordCounts = pairs.reduceByKey(_ + _);
```

Spark Streaming 有特定的窗口操作, 其涉及两个参数: 滑动窗口的宽度和频率。这两个参数必须为每个时间片的倍数。例如, 假设滑动窗口的宽度为 5 秒, 频率为 1 秒, 则会对过去 5 秒中每一秒的 WordCount 都进行统计, 再进行迭加得出 5 秒中的单词统计。如下所示:

```
val wordCounts = words.map(x => (x, 1)).reduceByKeyAndWindow(_ + _, Seconds(5s), seconds(1))
```

结果统计后, 将 Spark Streaming 的结果进行输出, 打印至屏幕和输入文件, 如下所示:


```
wordCount.print();
```

将 WordCount 的 DStream WordCounts 输入 HDFS 文件。

```
wordCounts = saveAsHadoopFiles("WordCount");
```

4. 启动 Spark Streaming

之前所做的所有步骤只是创建了执行流程，程序没有真正连接上数据源，也没有对数据进行任何操作，只是设定好了所有的执行计划，当 `ssc.start()` 启动后，Spark Streaming 才开始监听，收取数据，程序才真正进行所有预期的操作，如下所示：

```
ssc.start();
```

5. 退出 Spark Streaming

程序在计算完毕后退出现：

```
ssc.awaitTermination();
```

7.7 Spark 的安装

本节内容分为两部分：

- ① 在 CentOS 环境下，Spark 单机运行的安装和配置；
- ② 使用 Spark Shell 与 Spark 进行简单的交互。

7.7.1 单机运行 Spark

要深入地了解 Spark，实际的操作和实验是很有必要的。所以，搭建一个最简单的 Spark 单机运行环境是我们开始实践的第一步。

本书所采用的 Linux 环境为 CentOS 6.4 64bit。所使用的 Spark 版本为 1.1.0。

由于 Spark 是用 Scala 写的，Scala 是一个 JVM 上的语言，所以，Java 环境不可或缺。使用 yum 来安装 OpenJDK 以提供环境支持。

```
$sudo yum install java-1.7.0-openjdk
```

```
$sudo yum install java-1.7.0-openjdk-devel
```

打开 `/etc/profile` 配置 JAVA_HOME 环境变量：

```
$sudo vi /etc/profile
```

将如下信息添加到配置文件中：

```
export JAVA_HOME=JAVA_HOME=/usr/lib/jvm/java-1.7.0-openjdk-1.7.0.71.x86_64
export JRE_HOME=$JAVA_HOME/jre
export PATH=$PATH:$JAVA_HOME/bin
export CLASSPATH=.:$JAVA_HOME/lib/dt.jar:$JAVA_HOME/lib/tools.jar
```

保存并退出 vi (:wq)，并使用 source 指令使配置生效。

```
$sudo source /etc/profile
```

下载安装 Scala: Spark 1.1.0 所对应的 Scala 版本为 2.10.x，所以在选择 Scala 时需要注意与 Spark 要求的版本相对应。

```
$wget http://www.scala-lang.org/files/archive/scala-2.10.4.tgz
```

使用 tar 解压并剪切至/usr/lib/目录下。

```
$ tar -zxf scala-2.10.4.tgz
$ sudo mv scala-2.10.4 /usr/lib
```

打开/etc/profile，配置 SCALA_HOME 环境变量，保存并使改变生效。

```
$ sudo vi /etc/profile
# add the following lines at the end
export SCALA_HOME=/usr/lib/scala-2.10.4
export PATH=$PATH:$SCALA_HOME/bin

$ sudo source /etc/profile
```

验证配置是否正确：在命令行下运行 Java -version 和 Scala -version 分别检测 Java 和 Scala 是否已经就位（图 7.9）。

```
$java -version
$scala -version
```

```
[root@localhost home]# java -version
java version "1.7.0_71"
OpenJDK Runtime Environment (rhel-2.5.3.1.el6-x86_64 u71-b14)
OpenJDK 64-Bit Server VM (build 24.65-b04, mixed mode)
[root@localhost home]# scala -version
Scala code runner version 2.10.4 -- Copyright 2002-2013, LAMP/EPFL
[root@localhost home]#
```

图 7.9 验证 Java 和 Scala 的安装

Spark 官方提供了很多种不同的 Spark 预编译版本，使用了不同的 Hadoop 版本（Spark 会使用 Hadoop 中的 HDFS），包括 Hadoop 1.x，CDH4，Hadoop 2.3，Hadoop 2.4，MapR 3.x，MapR 4.x。本书我们选择了 Hadoop1.x 的预编译版本来演示 Spark 的使用。

在 <https://spark.apache.org/downloads.html> 下载页面中，Chose a package type 选项选择 Pre-built for Hadoop 1.x，并下载相应的包 spark-1.1.0-bin-hadoop1.tgz。完成下载后，即可解压并运行 Spark。

```
$tar -zxf spark-1.1.0-bin-hadoop1.tgz
```

验证安装：在解压完成后，我们即可运行在 bin 目录下的 spark-shell 来验证 Spark 是否能在主机上顺利运行（图 7.10）。

```
$/bin/spark-shell
```

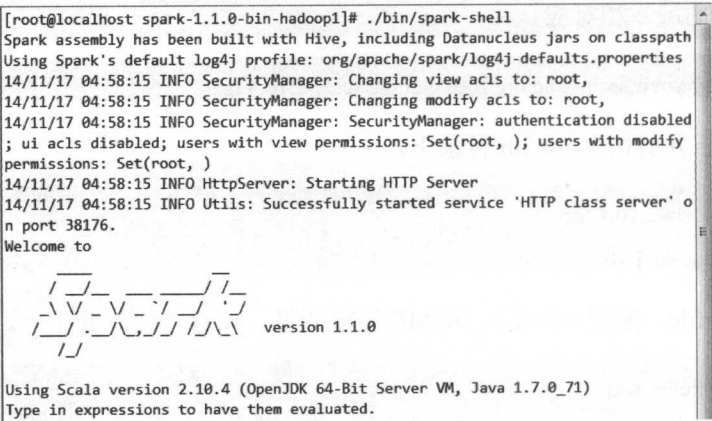


图 7.10 验证 Spark 安装

此时，Spark 的 Web UI 也已经运行起来了。可以在浏览器中输入安装 Spark 的主机的地址加上默认端口号 4040 去访问，若为本机，则地址应输入 <http://localhost:4040>（图 7.11）。

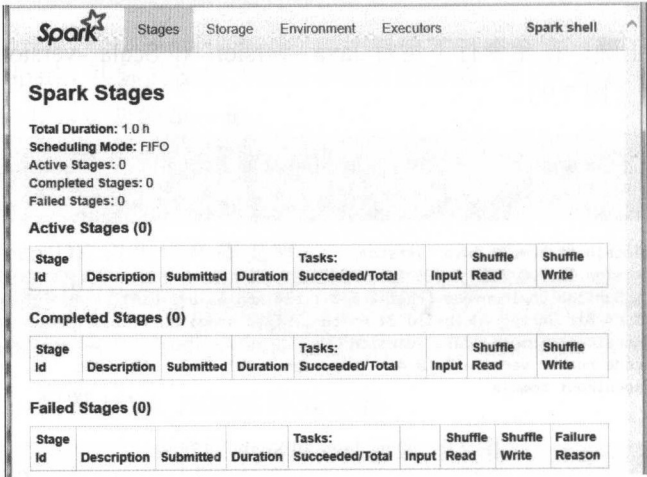


图 7.11 Spark UI

7.7.2 使用 Spark Shell 与 Spark 交互

Spark Shell 是一个特别适合让用户在 Spark 上快速试验原型的工具。通过它，可以与 Spark 集群进行交互，提交查询，这便于调试，也便于初学者使用 Spark。

与上一节一样，我们通过 `./bin/spark-shell` 连接到 Spark 集群上。

首先，我们看一下 Shell 环境下的 `sc` 变量，它是 Spark Shell 启动时自动生成的环境变量（图 7.12）。

```
14/11/17 22:43:46 INFO SparkUI: Started SparkUI at http://192.168.1.65:4040
14/11/17 22:43:47 INFO Executor: Using REPL class URI: http://192.168.1.65:41610
14/11/17 22:43:47 INFO AkkaUtils: Connecting to HeartbeatReceiver: akka.tcp://sparkDriver@
14/11/17 22:43:47 INFO SparkILoop: Created spark context..
Spark context available as sc.

scala> sc
res0: org.apache.spark.SparkContext = org.apache.spark.SparkContext@599f1a28

scala> █
```

图 7.12 `sc` 环境变量

可以看出 `sc` 就是 `SparkContext` 的实例，`SparkContext` 把代码提交到集群或者本地的通道，我们编写 Spark 代码，无论是要运行本地还是集群都必须有 `SparkContext` 的实例。

接下来，我们读取本地的 `README.md` 文件，对它进行一些实验性的操作（图 7.13）。

```
val inFile = sc.textFile("README.md")
```

```
scala> val inFile = sc.textFile("README.md")
14/11/17 22:51:29 INFO MemoryStore: ensureFreeSpace(32768) called with curMem=65536,
maxMem=280248975
14/11/17 22:51:29 INFO MemoryStore: Block broadcast_2 stored as values in memory (es
timated size 32.0 KB, free 267.2 MB)
inFile: org.apache.spark.rdd.RDD[String] = README.md MappedRDD[5] at textFile at <co
nsole>:12

scala> █
```

图 7.13 载入 `README.md` 文件

我们把读取的内容保存到了 `inFile` 这个变量，这是一个 `MappedRDD`，在 Spark 的代码编写中，一切都是基于 RDD 操作的。

然后我们从读取的文件中过滤出所有的包含 Spark 这个词的行（图 7.14）。

```
val sparks = inFile.filter(line => line.contains("Spark"))
```

```
scala> val sparks = inFile.filter(line => line.contains("Spark"))
sparks: org.apache.spark.rdd.RDD[String] = FilteredRDD[6] at filter at <console>:14

scala> █
```

图 7.14 过滤出含有 Spark 的行

最后，我们统计一下 Spark 一共出现在了多少行里（图 7.15）。

```
sparks.count
```

```
scala> sparks.count
14/11/17 22:58:52 WARN NativeCodeLoader: Unable to load native-hadoop library for yo
ur platform... using builtin-java classes where applicable
14/11/17 22:58:52 WARN LoadSnappy: Snappy native library not loaded
14/11/17 22:58:52 INFO FileInputFormat: Total input paths to process : 1
14/11/17 22:58:52 INFO SparkContext: Starting job: count at <console>:17
14/11/17 22:58:52 INFO DAGScheduler: Got job 0 (count at <console>:17) with 1 output
partitions (allowLocal=false)
14/11/17 22:58:52 INFO DAGScheduler: Final stage: Stage 0(count at <console>:17)
14/11/17 22:58:52 INFO DAGScheduler: Parents of final stage: List()
14/11/17 22:58:52 INFO DAGScheduler: Missing parents: List()
14/11/17 22:58:52 INFO DAGScheduler: Submitting Stage 0 (FilteredRDD[6] at filter at
<console>:14), which has no missing parents
14/11/17 22:58:53 INFO MemoryStore: ensureFreeSpace(2592) called with curMem=98304,
maxMem=280248975
14/11/17 22:58:53 INFO MemoryStore: Block broadcast_3 stored as values in memory (es
timated size 2.5 KB, free 267.2 MB)
14/11/17 22:58:53 INFO DAGScheduler: Submitting 1 missing tasks from Stage 0 (Filter
edRDD[6] at filter at <console>:14)
14/11/17 22:58:53 INFO TaskSchedulerImpl: Adding task set 0.0 with 1 tasks
14/11/17 22:58:53 INFO TaskSetManager: Starting task 0.0 in stage 0.0 (TID 0, localh
ost, PROCESS_LOCAL, 1201 bytes)
14/11/17 22:58:53 INFO Executor: Running task 0.0 in stage 0.0 (TID 0)
14/11/17 22:58:53 INFO HadoopRDD: Input split: file:/home/spark-1.1.0-bin-hadoop1/RE
ADME.md:0+4811
14/11/17 22:58:53 INFO Executor: Finished task 0.0 in stage 0.0 (TID 0). 1731 bytes
result sent to driver
14/11/17 22:58:53 INFO DAGScheduler: Stage 0 (count at <console>:17) finished in 0.1
60 s
14/11/17 22:58:53 INFO TaskSetManager: Finished task 0.0 in stage 0.0 (TID 0) in 153
ms on localhost (1/1)
14/11/17 22:58:53 INFO SparkContext: Job finished: count at <console>:17, took 0.558
170432 s
res3: Long = 21

scala> 14/11/17 22:58:53 INFO TaskSchedulerImpl: Removed TaskSet 0.0, whose tasks ha
ve all completed, from pool
```

图 7.15 统计行数

至此，我们通过 Spark Shell 完成了一次与 Spark 集群的简单交互。

7.8 Shark、Impala、Hive 对比

Berkeley AMP Lab 在 2014 年 2 月发布了一个针对 Shark、Impala、Hive 等大数据分析框架的评测标准——Big Data Benchmark，并对现有的典型大数据分析框架做了定量和定性的测试对比。参与测试对比的系统如下。

Redshift: 由 Amazon 出品的快速、完全托管的 PB 级数据仓库解决方案。

Hive: 基于 Hadoop 的数据仓库系统。

Shark: 基于 Spark 的 SQL 引擎，兼容 Hive。

Impala: 兼容 Hive 的类 MPP 架构的 SQL 查询引擎。

Stinger/Tez: 下一代基于 Hadoop 的执行引擎。

测试所使用的数据集基于 Pavlo 等人开发的 Benchmark，使用了与 Pavlo 的 Benchmark 相同的 Schema 和查询语句。但使用了不同的数据，这些数据由 Intel's Hadoop benchmark tools 生成，数据采样自 Common Crawl 文档语料库，共 3 个不同的数据集，见表 7.3。

表 7.3 数据集

Documents	Rankings	UserVisits
非结构化的 HTML 文档	网站列表及各个网站的 Page Rank	各个网页的服务器日志
	pageURL VARCHAR(300) pageRank INT avgDuration INT	sourceIP VARCHAR(116) destURL VARCHAR(100) visitDate DATE adRevenue FLOAT userAgent VARCHAR(256) countryCode CHAR(3) languageCode CHAR(6) searchWord VARCHAR(32) duration INT

所执行的查询测试如下。

Query 1: 扫描查询（Scan Query），如图 7.16 所示。

SELECT pageURL, pageRank FROM rankings WHERE pageRank > X

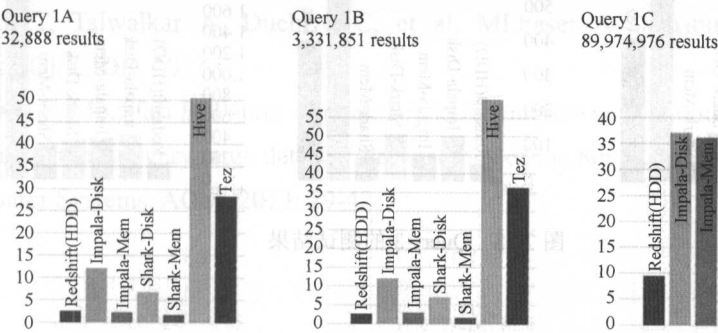


图 7.16 Query 1 的测试结果

Query2: 聚合查询（Aggregation Query），如图 7.17 所示。

SELECT SUBSTR(sourceIP, 1, X), SUM(adRevenue) FROM uservisits GROUP BY SUBSTR(sourceIP, 1, X)

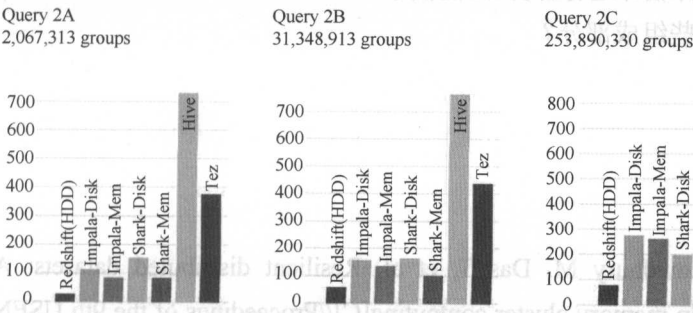
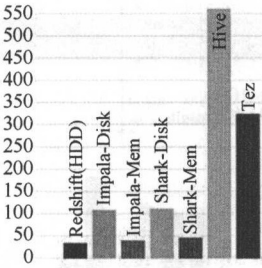


图 7.17 Query 2 的测试结果

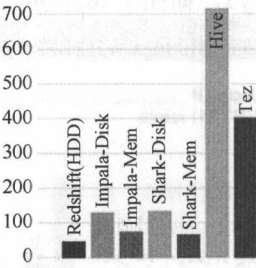
Query 3: Join 查询，如图 7.18 所示。

```
SELECT sourceIP, totalRevenue, avgPageRank
FROM
  (SELECT sourceIP,
    AVG(pageRank) as avgPageRank,
    SUM(adRevenue) as totalRevenue
  FROM Rankings AS R, UserVisits AS UV
  WHERE R.pageURL = UV.destURL
    AND UV.visitDate BETWEEN Date('1980-01-01') AND Date('X')
  GROUP BY UV.sourceIP)
ORDER BY totalRevenue DESC LIMIT 1
```

Query 3A
485,312 rows



Query 3B
53,332,015 rows



Query 3C
533,287,121 rows

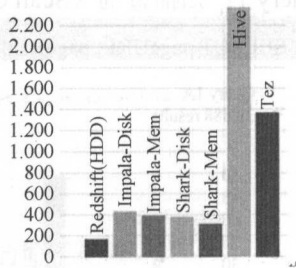


图 7.18 Query 3 的测试结果

7.9 练习题

1. Spark 和 Hadoop 的本质区别是什么？
2. RDD 在 Spark 中起到的关键作用是什么？
3. 部署 Spark 并编译运行计算 PI 的例子。
4. BDAS 有哪些组成部分？

参考文献

- [1] Zaharia M, Chowdhury M, Das T, et al. Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing[C]//Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation. USENIX Association, 2012: 2-2.

- [2] Zaharia M, Chowdhury M, Franklin M J, et al. Spark: cluster computing with working sets[C]//Proceedings of the 2nd USENIX conference on Hot topics in cloud computing. 2010: 10-10.
- [3] Xin R S, Gonzalez J E, Franklin M J, et al. Graphx: A resilient distributed graph system on spark[C]//First International Workshop on Graph Data Management Experiences and Systems. ACM, 2013: 2.
- [4] Zaharia M, Das T, Li H, et al. Discretized streams: Fault-tolerant streaming computation at scale[C]//Proceedings of the Twenty-Fourth ACM Symposium on Operating Systems Principles. ACM, 2013: 423-438.
- [5] Engle C, Lupher A, Xin R, et al. Shark: fast data analysis using coarse-grained distributed memory[C]//Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data. ACM, 2012: 689-692.
- [6] Li H, Ghodsi A, Zaharia M, et al. Tachyon: Reliable, Memory Speed Storage for Cluster Computing Frameworks[C]//Proceedings of the ACM Symposium on Cloud Computing. ACM, 2014: 1-15.
- [7] Kraska T, Talwalkar A, Duchi J C, et al. MLbase: A Distributed Machine-learning System[C]//CIDR. 2013.
- [8] Agarwal S, Mozafari B, Panda A, et al. BlinkDB: queries with bounded errors and bounded response times on very large data[C]//Proceedings of the 8th ACM European Conference on Computer Systems. ACM, 2013: 29-42.

第8章

Storm 流计算系统

Hadoop 等大数据解决方案解决了当今大部分对于海量数据的处理需求，如网页检索、机器翻译、分布式计算、广告投放等；但对于某些实时性要求很高的数据处理系统，Hadoop 则无能为力。对实时交互处理的需求催生了一大批实时计算系统，根据其数据流的特点，我们可以称之为流计算系统。本节主要介绍流计算的基本概念，以典型的流计算系统 Storm 为例深入分析流计算系统的特性。

8.1 流计算系统

8.1.1 流计算系统的特点

传统的数据处理流程，是先收集数据存放到数据库中，当有数据服务需要的时候，通过对数据库中的数据做一系列的查询和计算作为响应。从宏观上来看，这是一个被动的服务方式，且是非实时的。

而流计算可以很好地对大规模流动数据在不断变化的运动过程中实时地进行分析，捕捉到可能有用的信息，并把结果反馈给下一个计算节点。数据是流式的，计算与服务也是流式不间断的，整个过程是连续的，其响应也是实时的，可以达到秒级别以内。

简单概括一下流计算的主要特点：

- ① 流式数据。
- ② 实时计算。

1. 流式数据

仅从字面意义，我们可以理解，数据像水一样连续不断地流过。数据流计算来自一个信念：数据的价值随着时间的流逝而降低，所以事件出现后必须尽快地对它们进行处理，最好数据出现时便立刻对其进行处理，发生一个事件进行一次处理，而不是缓存起来成为一批后处理。在数据流模型中，需要处理的输入数据（全部或部分）并不存储在可随机访问的磁盘或内存中，它们以一个或多个“连续数据流”的形式到达。数据流不同于传统的存储关系模型，主要有以下几个方面的特点。

- ① 流中的数据元素在线到达，需要实时处理。
- ② 系统无法控制将要处理的新到达的数据元素的顺序，无论这些数据元素是在一个数据流中还是跨多个数据流，即重放的数据流可能和上次数据流的元素顺序不一致。
- ③ 数据流也许是无穷无尽的。
- ④ 一旦数据流中的某个元素经过处理，要么被丢弃，要么被归档存储。因此，除非该数据被直接存储在内存中，否则将不容易被检索。
- ⑤ 数据流系统涉及的操作分为有状态和无状态两种，无状态的算子包括 `union`、`filter` 等，有状态的算子包括 `bsort`、`join`、`aggregate` 等。有状态的算子如果执行失败后，其保持的状态会丢失，重放数据流产生的状态和输出不一定和失效前保持一致，而无状态的算子失败后，重放数据流能够构建与之前一致的输出。
- ⑥ 数据流计算可以看成是一个个算子（节点）和一条条数据流（边）组成的数据流图。

2. 实时计算

实时计算最核心的需求即响应时间为秒级以内。在很多实时应用场景中，比如实时交易系统、实时诈骗分析、实时广告推送、实时监控、社交网络实时分析等，数据量大，实时性要求高，而且数据源是实时不间断的。新到的数据必须马上处理完，不然后续的数据就会堆积起来，永远也处理不完。反应时间经常要求在秒级以下，甚至是毫秒级，这就需要有一个高度可扩展的流式计算解决方案。

实时计算就是针对实时连续的数据类型而准备的。在流数据不断变化的运动过程中实时地进行分析，捕捉到可能对用户有用的信息，并把结果发送出去。整个过程中，数据分析处理系统是主动的，用户却处于被动接收的状态。

8.1.2 流计算处理基本流程

流计算系统的基本处理流程可分为数据采集、数据计算、数据存储及服务。与其他数据处理系统不同的是，这几个阶段里，数据的采集，计算都是实时的。

实时数据采集：互联网企业的海量数据采集工具，有 Facebook 开源的 Scribe、LinkedIn 开源的 Kafka、Cloudera 开源的 Flume、淘宝开源的 TimeTunnel、Hadoop 的 Chukwa 等，

均可以满足每秒数百 MB 的日志数据采集和传输需求。

数据实时计算：传统的数据操作，首先将数据采集并存储在 DBMS 中，然后通过查询和 DBMS 进行交互。整个过程 DBMS 系统是被动的。而流计算的模式是：接收数据采集系统源源不断发来的实时数据后，流计算系统在流数据不断变化的运动过程中实时地进行计算分析，实时提供数据服务。流式数据处理如图 8.1 所示。

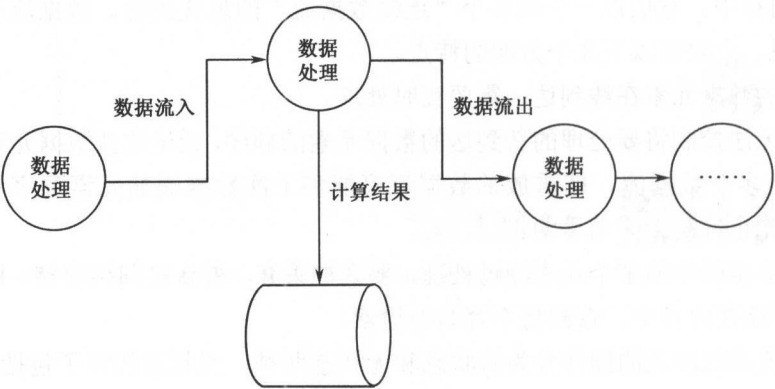


图 8.1 流式数据处理

数据存储及服务：理想情况下，流计算会将对用户有价值的结果实时推送给用户，这取决于应用场景。一般而言，流计算的第三个阶段是实时查询服务，经由流计算框架得出的结果可供用户进行实时查询、展示或存储。

8.2 Storm 流计算框架

8.2.1 Storm 简介

Storm 是 Twitter 的一个分布式、容错的实时计算系统，如今已经正式开源。Storm 可以方便地在一个计算机集群中编写与扩展复杂的实时计算。Storm 定义了一批实时计算的原语，如同 Hadoop 大大简化了并行批量数据处理，Storm 的这些原语大大简化了并行实时数据处理。按照 Storm 作者的话，Storm 之于实时处理，就好比 Hadoop 之于批处理。Storm 保证每个消息都会得到处理，而且它很快在一个小集群中，每秒可以处理数以百万计的消息，而且可以使用任意编程语言来开发。

Storm 的一些关键特性如下。

① 适用场景广泛：Storm 可以用来处理消息和更新数据库（消息流处理），对一个数据量进行持续的查询并返回客户端（持续计算），对一个耗资源的查询做实时并行化的处理（分布式方法调用），Storm 的这些基础原语可以满足大量的场景。

② 可伸缩性高：Storm 的可伸缩性可以让 Storm 每秒可以处理的消息量达到很高。为了扩展一个实时计算任务，所需要做的就是增加机器并且提高这个计算任务的并行度设置 (parallelism setting)。

③ 保证无数据丢失：实时系统必须保证所有的数据被成功地处理。那些会丢失数据的系统的适用场景非常窄，而 Storm 保证每一条消息都会被处理，这一点和 S4 相比有巨大的反差。

④ 异常健壮：Storm 集群非常容易管理，这是 Storm 的设计目标之一。

⑤ 容错性好：如果在消息处理过程中出了一些异常，Storm 会重新安排这个出问题的处理逻辑。Storm 保证一个处理逻辑永远运行，除非显式杀掉这个处理逻辑。

⑥ 语言无关性：健壮性和可伸缩性不应该局限于一个平台。Storm 的 Topology 和消息处理组件可以用任何语言来定义，这一点使得任何人都可以使用 Storm。

8.2.2 Storm 关键技术语

Storm 在设计实现中抽象出了很多概念，图 8.2 为 Storm 数据流图，也包含了 Storm 的几个关键概念。

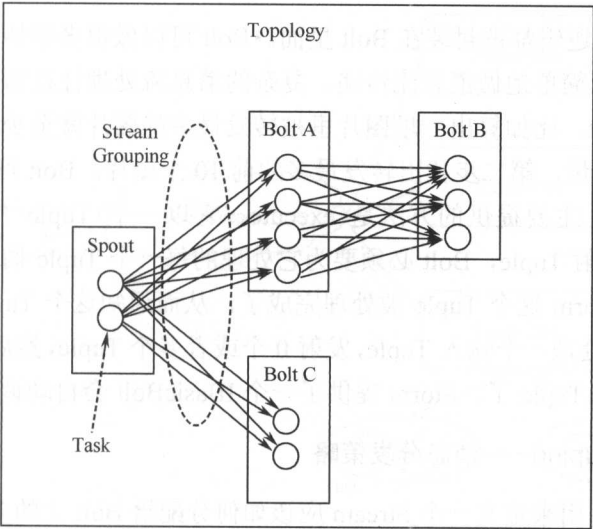


图 8.2 Storm 数据流图

其中的关键技术语包括 Topology、Spout、Bolt、Task、Stream Grouping。

1. Stream——流

流是 Storm 最核心的一个抽象。一个流是一个没有边界的 Tuple 序列，而这些 Tuple 会被以一种分布式的方式并行地创建和处理。对消息流的定义主要是对消息流的 Tuple 的

定义,我们会给 Tuple 的每个字段起一个名字,并且不同 Tuple 的对应字段的类型必须一样。也就是说,两个 Tuple 的第一个字段的类型必须一样,第二个字段的类型必须一样,但是第一个字段和第二个字段可以有不同的类型。

2. Tuple——消息单元、元组

它是消息传递的基本单元。本来应该是一个 key-value 的 Map,但是由于各个组件间传递的 Tuple 的字段名称已经事先定义好,所以 Tuple 中只要按序填入各个 value 就行了,所以就是一个 value list。

3. Spout——源数据流

它也可以称为消息源,Spout 是 Storm 一个 Topology 的数据生产者。一般来说,消息源会从一个外部源读取数据并且向 Topology 里面发出消息 Tuple。消息源 Spout 可以是可靠的也可以是不可靠的。一个可靠的消息源可以重新发射一个 Tuple,如果这个 Tuple 没有被 Storm 成功地处理,但是一个不可靠的消息源 Spout 一旦发出,一个 Tuple 就把它彻底忘了,也就不可能再发了。消息源可以发射多条消息流 Stream。Spout 是主动的,其接口中的 nextTuple()函数会不断接收 Storm 框架的调用。

4. Bolt——消息处理者

所有的消息处理逻辑都被封装在 Bolt 里面。Bolt 可以做很多事情:过滤、聚合、查询数据库等。Bolt 可以简单地做消息流传递。复杂的消息流处理往往需要很多步骤,从而也就需要经过很多 Bolt。比如算出一堆图片里被转发最多的图片就至少需要两步:第一步算出每个图片的转发数量,第二步找出转发最多的前 10 张图片。Bolt 可以发射多条消息流。

Bolt 是被动的,主要提供的方法是 execute,它以一个 Tuple 作为输入,Bolt 使用 OutputCollector 来发射 Tuple,Bolt 必须要为它处理的每一个 Tuple 调用 OutputCollector 的 ack 方法,以通知 Storm 这个 Tuple 被处理完成了,从而通知这个 Tuple 的发射者 Spout。一般的流程是:Bolt 处理一个输入 Tuple,发射 0 个或者多个 Tuple,然后调用 ack 通知 Storm 自己已经处理过这个 Tuple 了。Storm 提供了一个 IBasicBolt 会自动调用 ack。

5. Stream Grouping——消息分发策略

Stream Grouping 用来定义一个 Stream 应该如何分配给 Bolt 上的多个 Task。Storm 有 6 种类型的 Stream Grouping。

① Shuffle Grouping: 随机分组,随机派发 Stream 里面的 Tuple,保证每个 Bolt 接收到的 Tuple 数目相同。

② Fields Grouping: 按字段分组,比如按 userid 来分组,具有同样 userid 的 Tuple 会被分到相同的 Bolt,而不同的 userid 则会被分配到不同的 Bolt。

③ All Grouping: 广播发送,对于每一个 Tuple,所有的 Bolt 都会收到。

④ Global Grouping: 全局分组,这个 Tuple 被分配到 Storm 中的一个 Bolt 的其中一个

Task。再具体一点就是分配给 ID 值最低的那个 Task。

⑤ Non Grouping: 不分组, Stream 不关心到底谁会收到它的 Tuple。目前这种分组和 Shuffle Grouping 是一样的效果,有一点不同的是 Storm 会把这个 Bolt 放到这个 Bolt 的订阅者的线程里执行。

⑥ Direct Grouping: 直接分组,这是一种比较特别的分组方法,用这种分组意味着消息的发送者指定由消息接收者的哪个 Task 处理这个消息。

8.2.3 Storm 架构设计

Storm 主要由 Nimbus, ZooKeeper 和 Supervisor 这三大组件组成。Nimbus 和 Supervisor 都是快速失败 (fail-fast)、无状态的,这样它们就变得十分健壮,两者的协调工作是由 ZooKeeper 来完成的。ZooKeeper 用于管理集群中的不同组件。任务状态和心跳信息等都保存在 ZooKeeper 上,提交的代码资源都在本地机器的硬盘上。

与 Hadoop 一样,Storm 也是 Master-Slave 架构,集群由一个主节点和多个工作节点组成。主节点运行 Nimbus 守护进程,每个工作节点都运行了一个 Supervisor 守护进程,用于监听工作,开始并终止工作进程。Storm 的组件架构如图 8.3 所示。

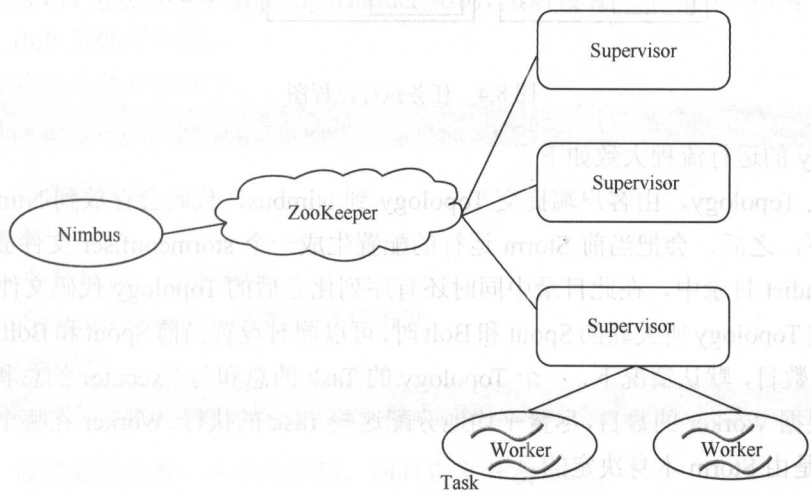


图 8.3 Storm 组件架构

下面详细介绍一下各个组件的功能。

- ① Nimbus: 发布分发代码, 分配任务, 监控状态。
- ② Supervisor: 监听宿主节点, 接受 Nimbus 分配的任务, 根据需要启动/关闭工作进程 Worker。
- ③ ZooKeeper: Storm 重点依赖的外部资源。Nimbus、Supervisor 和 Worker 都把心跳保存在 ZooKeeper 上。Nimbus 也是根据 ZooKeeper 上的心跳和任务运行状况, 进行调度和

任务分配的。

④ Worker: 运行具体处理组件逻辑的进程。

⑤ Task: Worker 中每一个 Spout/Bolt 的线程称为一个 Task。在 Storm 0.8 之后, Task 不再与物理线程对应, 同一个 Spout/Bolt 的 Task 可能会共享一个物理线程, 该线程称为 Executor。

图 8.4 是 Topology 提交的流程示意图, 从这个执行流程我们可以很清晰地看到 Storm 各个组件之间的功能与交互关系。

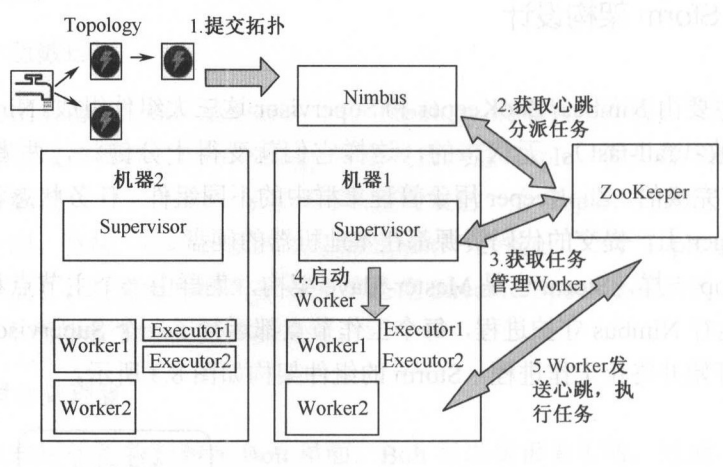


图 8.4 任务执行流程图

Topology 的运行流程大致如下。

① 定义 Topology, 由客户端提交 Topology 到 Nimbus, 代码会存放到 Nimbus 节点的 inbox 目录下, 之后, 会把当前 Storm 运行的配置生成一个 stormconf.ser 文件放到 Nimbus 节点的 stormdist 目录中, 在此目录中同时还有序列化之后的 Topology 代码文件。

② 设定 Topology 所关联的 Spout 和 Bolt 时, 可以同时设置当前 Spout 和 Bolt 的 Executor 数目和 Task 数目, 默认情况下, 一个 Topology 的 Task 的总和与 Executor 的总和是一致的。之后, 系统根据 Worker 的数目, 尽量平均地分配这些 Task 的执行。Worker 在哪个 Supervisor 节点上运行是由 Storm 本身决定的。

③ 任务分好后, Nimbus 将任务的信息提交到 ZooKeeper 集群中, 同时在 ZooKeeper 集群中会有 worerbeats 节点, 这里存储了当前 Topology 的所有 Worker 进程的心跳信息。

④ Supervisor 获取所分配的任务, 启动任务; Supervisor 节点会不断地轮询 ZooKeeper 集群, 在 ZooKeeper 的 assignments 节点中保存了所有 Topology 的任务分配信息、代码存储目录、任务之间的关联关系等, Supervisor 通过轮询此节点的内容, 来领取自己的任务, 启动 Worker 进程运行。

⑤ Worker 节点中的 Task 执行具体的任务逻辑, 并且实时给 ZooKeeper 发送心跳状态信息。

说明：一个 Topology 运行之后，就会不断地通过 Spout 来发送 Stream 流，通过 Bolt 来不断地处理接收到的 Stream 流，Stream 流是无界的。最后一步会不间断地执行，除非手动结束 Topology。

8.3 Storm 编程实例

下面介绍大家熟悉的 WordCount 编程实例，分析其源码，以加深对 Storm 编程及其设计思想的理解。WordCount 的实例代码来源于 Storm 官方 github，编程语言环境为 Java。

1. 创建一个简单的 Topology

Topology 定义了整个计算逻辑，代码如下所示：

```
TopologyBuilder builder = new TopologyBuilder();
builder.setSpout("spout", new RandomSentenceSpout(), 5);
builder.setBolt("split", new SplitSentence(), 8).shuffleGrouping("spout");
builder.setBolt("count", new WordCount(), 12).fieldsGrouping("split", new Fields("word"));
```

总体来说，首先创建一个 TopologyBuilder 实例，然后设置一个 Spout 和两个 Bolt。Spout 是数据源，Bolt 是处理单元。

每一行代码具体介绍如下。

```
builder.setSpout("spout", new RandomSentenceSpout(), 5);
```

作用：数据源设置。

参数列表：

Spout，名字标识，数据来源。

RandomSentenceSpout()，数据源处理方法。

5，并发线程数。

```
builder.setBolt("split", new SplitSentence(), 8).shuffleGrouping("spout");
```

作用：设置处理函数——单词分割，同时定义分发策略。

参数列表：

split，Bolt 的名字标识，单词的分割。

SplitSentence()，单词分割函数。

shuffleGrouping，分发策略为随机，其数据来源为上一行的 Spout。

```
builder.setBolt("count", new WordCount(), 12).fieldsGrouping("split", new Fields("word"));
```

作用：设置处理函数——计数，同时定义分发方式为按字段分组，只有具有相同 field 值的 Tuple 才会发给同一个 Task 进行统计，保证了统计的准确性。

参数列表:

count, 定义 Bolt 的名字标识为单词计数。

WordCount(), 具体的计数函数。

fieldsGrouping, 分发策略为按字段分组, 数据来源为上一行定义的 split, 定义了一个新的字段。

2. 写具体的函数实现

SplitSentence——单词分割函数的实现:

```
public static class SplitSentence extends ShellBolt implements IRichBolt {
```

```
    public SplitSentence() {
```

```
        super("python", "splitsentence.py");
```

```
    }
```

```
    @Override
```

```
    public void declareOutputFields(OutputFieldsDeclarer declarer) {
```

```
        declarer.declare(new Fields("word"));
```

```
    }
```

```
    @Override
```

```
    public Map<String, Object> getComponentConfiguration() {
```

```
        return null;
```

```
    }
```

```
}
```

super("python", "splitsentence.py"); 调用外置的 Python 代码来实现, 脚本是单词分割的具体实现。

SplitSentence.py——真正的分割函数:

```
import storm
```

```
class SplitSentenceBolt(storm.BasicBolt):
```

```
    def process(self, tup):
```

```
        words = tup.values[0].split(" ")
```

```
        for word in words:
```

```
            storm.emit([word])
```

```
SplitSentenceBolt().run()
```

分割后的单词通过 emit 的方法将 Tuple 发射出去，以便订阅了该 Tuple 的 Bolt 接收。

WordCount——词频统计函数的实现：

```
public static class WordCount extends BaseBasicBolt {
    Map<String, Integer> counts = new HashMap<String, Integer>();
    @Override
    public void execute(Tuple tuple, BasicOutputCollector collector) {
        String word = tuple.getString(0);
        Integer count = counts.get(word);
        if (count == null)
            count = 0;
        count++;
        counts.put(word, count);
        collector.emit(new Values(word, count));
    }

    @Override
    public void declareOutputFields(OutputFieldsDeclarer declarer) {
        declarer.declare(new Fields("word", "count"));
    }
}
```

WordCount 需要重写 BaseBasicBolt 类中的 execute 方法，execute 是 Bolt 的执行逻辑，后面 declareOutputFields 方法定义了最终的输出字段：("word", "count")。

3. 定义配置及任务提交

```
Config conf = new Config();
conf.setDebug(true);
if (args != null && args.length > 0) {
    conf.setNumWorkers(3);
    StormSubmitter.submitTopology(args[0], conf, builder.createTopology());
}
else {
    conf.setMaxTaskParallelism(3);
    LocalCluster cluster = new LocalCluster();
    cluster.submitTopology("word-count", conf, builder.createTopology());
    Thread.sleep(10000);
    cluster.shutdown();
}
```

这里定义了 Storm 的基本配置，例如 Debug 模式、设置 Worker 的数目，以及提交拓扑任务等。

8.4 Storm 应用

8.4.1 Storm 应用场景

由于其在实时计算方面的优势，Storm 这种高可拓展性、能处理高频数据和大规模数据的实时流计算解决方案将被应用于实时搜索、高频交易和社交网络等。金融机构的交易系统也是一个典型的流计算处理系统，它对系统的实时性和一致性有很高要求。

Storm 的所有者 Twitter 列举了 Storm 的三大作用领域。

1. 信息流处理（Stream Processing）

Storm 可以用来实时处理新数据和更新数据库，兼具容错性和可扩展性。

2. 连续计算（Continuous Computation）

Storm 可以进行连续查询并把结果即时反馈给客户，比如将 Twitter 上的热门话题发送到客户端。

3. 分布式远程过程调用（Distributed RPC）

Storm 可以用来并行处理密集查询，Storm 的拓扑结构是一个等待调用信息的分布函数，当它收到一条调用信息后，会对查询进行计算，并返回查询结果。

8.4.2 Storm 应用实例

在淘宝，Storm 被广泛用来进行实时日志处理，出现在实时统计、实时风控、实时推荐等场景中。一般来说，淘宝从类 Kafka 的 metaQ 或者基于 HBase 的 Time Tunnel 中读取实时日志消息，经过一系列处理，最终将处理结果写入一个分布式存储，提供给应用程序访问。每天的实时消息量从几百万到几十亿不等，数据总量达到 TB 级。对于我们来说，Storm 往往会配合分布式存储服务一起使用。在一个正在进行的个性化搜索实时分析项目中，就使用了 Time Tunnel + HBase + Storm + UPS 的架构，每天处理几十亿的用户日志信息，从用户行为发生到完成分析延迟在秒级。

在腾讯，腾讯应急安全中心基于 Storm 开发了一套 CGI 采集与清理系统——Storm-CGI。CGI 好比 Web 漏洞扫描器的眼睛，只有 CGI 更全更准，Web 漏洞扫描器才能更好地“看到”漏洞，为业务的 Web 安全保驾护航。Storm-CGI 系统采集 CGI 的来源主要有三种，分别是

IDS 光纤旁路出来的 HTTP 请求日志文件、门神旁路的 HTTP 请求日志文件、还有 Web 2.0 爬虫抓取的 URL。Storm-CGI 中的 Spout 组件 Valid_Rewrite_Spout 从这些数据源中抓取 CGI，并进行合法性过滤和 Rewrite 过滤、HTTP 探测过滤，最终得到高质量的实际存在的 CGI。Storm-CGI 系统还能从 CGI 库中读取库存 CGI 数据，进行迭代过滤，保证库存 CGI 数据的准确有效性。Storm-CGI 系统的整体架构如图 8.5 所示。

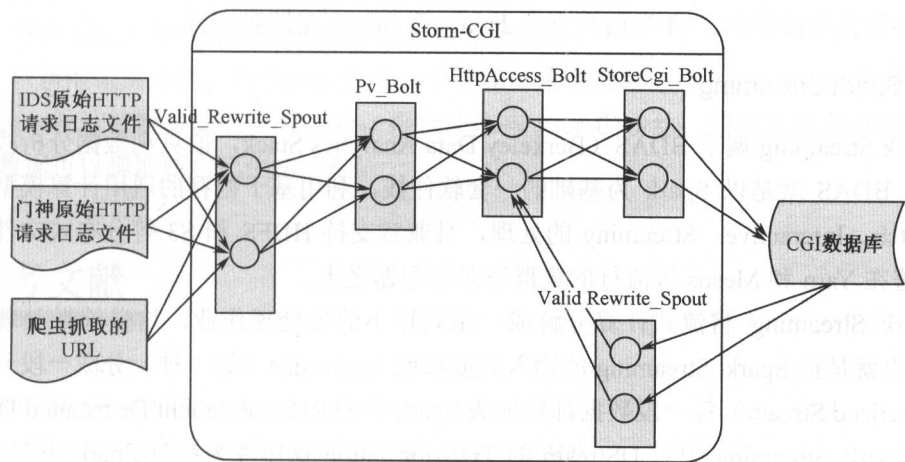


图 8.5 腾讯 Storm-CGI 系统的整体架构

目前，Storm-CGI 是由分布在不同 IDC 机房的 13 台机器组成的小分布式集群，每天可处理 2TB 左右的日志文件，每天平均过滤 4 亿个 CGI 数据，从中采集到 5 万左右准确的 CGI（部分 CGI 在 CGI 库中已经存在）。

Storm-CGI 能从大量的数据中实时地采集出海量的 CGI 数据，并通过合法性过滤、Rewrite 过滤、HTTP 探测过滤，最终得到准确的 CGI 数据，供 Web 漏洞扫描器做安全漏洞扫描。它好比 Web 漏洞扫描器的眼睛，能让 Web 漏洞扫描器透过海量的 URL 数据，看到真实准确的 CGI，从而发现 Web 安全漏洞。

8.5 其他流计算框架

除了 Storm 还有一些比较流行的流计算框架，如 Yahoo S4，Spark Streaming，Facebook Puma 等。下面以 Yahoo S4 和 Spark Streaming 为例，介绍与 Storm 设计上的异同点。

1. Yahoo S4

S4 是一个受 MapReduce 模式启发的分布式流处理引擎，从架构上来说，与 Storm 很大的一个不同点是 S4 的去中心化设计，其源于 MapReduce 和 Actor 模式的结合。因为其对称

的结构，S4 的设计非常简单。

S4 将事件流抽象为以 (key, value) 形式的元素组成的序列，这里 key 和 value 分别是键和属性。在这种抽象的基础上 S4 设计了能够消费和发出这些 (key,value) 元素的组件，也就是 Process Element PE。Process Element 在 S4 中是最小的数据处理单元，每个 PE 实例只消费属性 key、属性 value 都匹配的事件，并最终输出结果或者输出新的元素。而 Storm 提供 Tuple 类，用户不可以自定义事件类，但是可以命名 field 和注册序列化器。

在多语言支持方面，S4 当前只支持 Java，而 Storm 支持多语言。

2. Spark Streaming

Spark Streaming 属于 BDAS (Berkeley Data Analytics Stack, 伯克利数据分析栈) 中的一部分，BDAS 就是以 Spark 为基础的一套软件栈，利用基于内存的通用计算模型，同时支持 Batch、Interactive、Streaming 的处理，且兼容支持 HDFS 和 S3 等分布式文件系统，可以部署在 Yarn 和 Mesos 等流行的集群资源管理器之上。

Spark Streaming 将流式计算分解成一系列短小的批处理作业，这里的批处理引擎是 Spark，也就是把 Spark Streaming 的输入数据按照 batch size (如 1 秒) 分成一段一段的数据 (Discretized Stream)，每一段数据都转换成 Spark 中的 RDD (Resilient Distributed Dataset)，然后将 Spark Streaming 中对 DStream 的 Transformation 操作变为针对 Spark 中对 RDD 的 Transformation 操作，将 RDD 经过操作变成中间结果保存在内存中。整个流式计算根据业务的需求可以对中间的结果进行叠加，或者存储到外部设备中。

它与 Storm 的根本的区别有以下几点：

(1) 处理模型、延迟

Spark Streaming 将流式数据分成一系列短小的批处理作业，其处理的是某个时间窗口内的事件流。而 Storm 处理的是每次传入的一个事件，因此 Storm 处理一个事件的延迟在 1 秒以内，而 Spark 则有几秒的延迟。

(2) 容错、数据保证

Spark Streaming 在容错性方面提供了较好支持容错状态计算。在 Storm 中，每个单独的记录通过系统时必须被跟踪，所以 Storm 能够至少保证每个记录将被处理一次，但是在从错误中恢复过来时允许出现重复记录。这意味着可变状态可能不正确地被更新两次。

另一方面，Spark Streaming 只需要在批级别进行跟踪处理，因此可以有效地保证每个 mini-batch 将完全被处理一次，即便一个节点发生故障。

(3) 实现、编程 API

Spark Streaming 由 Scala 实现，而 Storm 最初是由 Clojure 实现的。但 Storm 和 Spark Streaming 都支持 Java API。Spark Streaming 运行于 Spark 之上，可以编写相同的代码来处理实时流数据和历史数据。

8.6 练习题

1. 分析流处理与批处理在系统架构设计上的异同点。
2. 简述 Storm 与 Yahoo S4 在计算流程上的分析与对比。
3. 对比 Storm 与 Spark Streaming 的集群容错机制。
4. 以新浪微博为例, 用 Storm 开发一个计算一条微博到达率 (Reach) 的程序。Reach 计算方法: 统计转发微博的所有 user, 再统计这些 user 的粉丝, 最后求出所有粉丝的 Set 集合。数据可以抓取或自己生成一些。

参考文献

- [1] <https://storm.apache.org/>
- [2] <https://storm.apache.org/documentation/Concepts.html>
- [3] Segerberg A, Bennett W L. Social media and the organization of collective action: Using Twitter to explore the ecologies of two climate change protests[J]. The Communication Review, 2011, 14(3): 197-215.
- [4] <http://tech.uc.cn/?p=2159>
- [5] <https://github.com/nathanmarz/storm-starter/blob/master/src/jvm/storm/starter/WordCountTopology.java>
- [6] <http://incubator.apache.org/s4/doc/0.6.0/>
- [7] <http://www.csdn.net/article/2014-01-27/2818282-Spark-Streaming-big-data>
- [8] <http://www.searchtb.com/2012/09/introduction-to-storm.html>
- [9] <http://security.tencent.com/index.php/blog/msg/39>

第9章

SQL、NoSQL 与 NewSQL

一直以来数据是基础也是核心，存储数据是前提，处理好数据是基础。之前企业一直用传统关系型数据库来存储数据，随着大数据时代的来临，需要存储的数据的量和数据的结构和种类都发生了很大的变化，人们已经不满足于传统关系型数据库带来的稳定性，对数据库支持的数据种类、伸缩性、可扩展性的需求越来越高，随着数据库技术的发展，NoSQL 应运而生，发展的时间虽然不长，但已经成为大数据技术生态圈里不可或缺的一员，NoSQL 的广泛应用也带动了新型数据库技术——NewSQL 的发展。本章由传统 SQL 数据库引出 NoSQL，重点介绍 NoSQL 的基本原理及其应用，最后概要介绍 NewSQL。

9.1 传统 SQL 数据库

几乎从一开始，在整个软件行业，传统关系型数据库是首选的存储解决方案。即使现在 NoSQL 技术发展迅速，关系型数据库在整个数据库领域的稳定地位还是不可取代的。本节主要介绍传统 SQL 数据库的设计思想以及最佳的应用场景，最后由传统 SQL 数据的不足引出 NoSQL 数据库。

9.1.1 关系模型

关系型数据库是建立在关系模型基础上的，关系模型是由埃德加·科德于 1970 年首先提出的，现如今虽然对此模型有一些批评意见，但它还是数据存储的传统标准。

借助于集合代数等数学概念和方法来处理数据库中的数据，用这种关系模型来表示现实世界中的各种实体以及实体之间的各种联系。在数据库系统中关系模型主要由关系数据结构、关系操作集合、关系完整性约束三部分组成。

简单来说，关系模型指的就是二维表格模型，而一个关系型数据库就是由二维表及其之间的联系所组成的一个数据组织。一个关系可以用一个表来表示，一个数据库可以包含多个表以表示多个关系。表由一系列的元组（行）组成，元组由列的值组成，数据库所有的操作都以元组为目标。可以认为传统关系型数据库是基于行的数据库，后面将提到的 NoSQL 数据库有基于列的模式。图 9.1 定义的表，简单地展示了基于行的关系模型。

用户信息表—user				
User				
user_id	name	age	class_id	元 组
1	Jack	22	201	

班级信息表—class		
class_id	grade	number
201	3	55

图 9.1 关系模型示例

SQL 就是一种基于关系数据库的查询语言，可以对关系数据库中的数据实现查询等操作。

9.1.2 关系型数据库的优点

即使 NoSQL 发展得如火如荼的今天，Twitter 还在用 MySQL 存储用户发的文章——“推文”，这些“推文”是 Twitter 最宝贵的数据资源。炒得很热的 NoSQL 数据库 Cassandra 最终也没能取代 MySQL。另外现在大部分的银行业务系统和传统企业 ERP 系统等都使用着稳定的关系型数据库，关系型数据库的优点如下。

- ① 容易理解：二维表结构是非常贴近逻辑世界的一个概念。
- ② 使用方便：通用的 SQL 语言使得操作关系型数据库非常方便，且支持比较复杂的查询情况。
- ③ 高性能：出色的索引，可以通过组合不同的查询高效地获得数据。
- ④ 事务的一致性：严格的强一致性支持。
- ⑤ 易于维护：丰富的完整性（实体完整性、参照完整性和用户定义的完整性）大大降低了数据冗余和数据不一致的概率。

传统关系型数据库的优势不容忽视，也是数据库技术的基石。但随着互联网技术的发展，它也面临了一些问题。

9.1.3 关系型数据库面临的问题

互联网的快速发展,也使得互联网上的数据得到爆发性的增长,累计的数据量越来越大,数据的类型也越来越丰富,随之带来的数据交互性能方面的问题也日益突出。

关系型数据库的最大特点就是对于事务和一致性的有力支持,传统的关系型数据库读写操作都是事务的,满足 ACID[原子性(Atomicity)、一致性(Consistency)、隔离性(Isolation)、持久性(Durability)]的特点,一致性是关系型数据库的灵魂(其他三个都是为其服务的),这个特性使得关系型数据库可以用于几乎所有对一致性有要求的系统中,如典型的银行系统。

数据库技术广泛应用于 Web 应用中,但在 Web 应用中,尤其是社交网络(SNS)应用中,一致性却并没有那么重要,很多时候短时间内的不一致性是可以容忍的。比如社交网站上用户 A 看到的和用户 B 看到的同一用户 C 内容更新不一致是可以容忍的,因此,关系型数据库的最大特点在当今最流行的应用中已经不是那么重要了。

关系型数据库为了维护一致性所付出的巨大代价就是其读写性能比较差,而像微博、Twitter 这类 SNS 的应用,对并发读写能力要求极高,关系型数据库已经无法应付。

关系数据库的另一个特点就是其具有固定的表结构,因此,其扩展性极差,而在社交网络中,系统的升级、功能的增加,往往意味着数据结构巨大改动,这一点关系型数据库也难以应付,需要新的结构化数据存储。

总结一下,针对于现在越来越复杂、丰富的应用,传统关系型数据库面临以下几个突出的问题:

- ① 数据库表结构不固定,可能会频繁变动;
- ② 对海量数据的高效率存储和访问的支持;
- ③ 针对弱一致性需求的优化。

9.2 NoSQL

NoSQL 可以写成 Not Only SQL,是对不同于传统的关联式数据库的数据库管理系统的统称,前面也提到过了, NoSQL 可以弥补传统关系数据库的不足,随着互联网 Web 应用的流行爆发,传统的关系数据库在应对这类应用时已经显得力不从心,非关系型的数据库则由于其本身的特点得到了非常迅速的发展。本节将详细介绍 NoSQL 与大数据的关系、NoSQL 的一些理论基础。

9.2.1 NoSQL 与大数据

为什么要用 NoSQL, NoSQL 跟大数据到底有什么关系, 最直白的解释就是 NoSQL 可以很轻松友好地应对大数据时代对大数据的存储以及交互需求。

大数据时代, 数据的显著特点就是数据量大, 这些数据是 TB 或 PB 级别以上的量级; 数据结构不统一, 包括结构化的、半结构化的和非结构化的数据, 其规模或复杂程度超出了常用传统数据库和软件技术所能管理和处理的数据集范围。

NoSQL 有良好、便捷的横向扩展性, 可以满足海量数据的存储需求。NoSQL 是一种无模式的数据存储模型, 可以应对 Web 应用上各种半结构化的数据, 灵活简单的数据模型以及弱一致性的特性使得高并发情况下数据查询的性能优异。可以说 NoSQL 是大数据时代数据库领域不可或缺的重要一员。

下面总结一下 NoSQL 的主要优势与特点。

灵活的数据模型: 多样的数据模型支持, 有基于 key-value 的、基于列存储的、基于图的一系列数据模型。

灵活的可扩展性、经济性: 相对于 RDBMS 来说, NoSQL 最突出的一个特点就是横向扩展, NoSQL 数据库通常使用廉价的服务器集群来管理膨胀的数据和事务数量, 而 RDBMS 通常需要依靠昂贵的专有服务器和存储系统来做到这一点。使用 NoSQL, 每 GB 的成本或每秒处理事务的成本, 都比使用 RDBMS 少很多倍, 可以花费更低的成本来存储和处理更多的数据。

9.2.2 NoSQL 理论基础

提到 NoSQL 不得不提 CAP 和最终一致性, CAP 是 NoSQL 的理论基石, 最终一致性是 NoSQL 很重要的一个特点。下面详细论述这两大理论。

1. CAP 理论

理解 CAP, 首先要明白什么是 CAP, 即 CAP 理论是什么? 其次要了解为什么, 即 CAP 理论对 NoSQL 为什么如此重要?

CAP 定理, 又称布鲁尔定理 (Brewer's theorem), 它指出对于一个分布式计算系统来说, 不可能同时满足以下三点。

- ① 一致性 (Consistency): 所有节点访问同一份最新的数据副本。
- ② 可用性 (Availability): 对数据更新具备高可用性, 每个请求不管成功或者失败都有响应。
- ③ 分区容错性 (Partition tolerance): 以实际效果而言, 分区相当于对通信的时限要求。系统如果不能在时限内达成数据一致性, 就意味着发生了分区的情况, 必须就当前操作在一致性和可用性之间做出选择。

在 2002 年，麻省理工学院的 Seth Gilbert 和 Nancy Lynch 发表了布鲁尔猜想的证明，使 CAP 成为了一个定理。

通俗地说，一致性、可用性、分区容错性三者是不能同时满足的，所以为了高度的可用性，只能牺牲一致性，以最终一致性取而代之，这颠覆了传统关系数据库的理论基础，CAP 理论使人豁然开朗。

为什么说它对 NoSQL 很重要？NoSQL 的设计理论基础即为 CAP 理论，NoSQL 设计之前就要在 CAP 上做取舍。通过图 9.2 我们可以看到不同的 NoSQL 数据库对 CAP 的取舍不同。

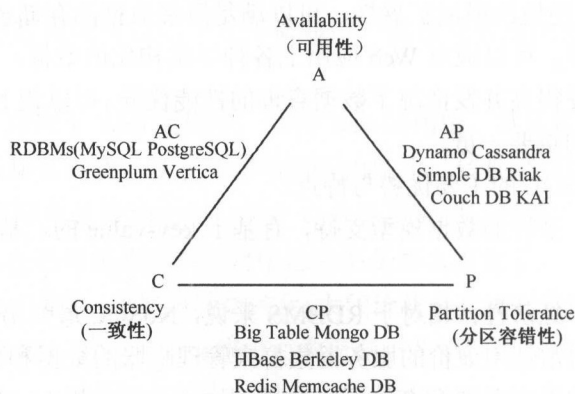


图 9.2 NoSQL 与 CAP

这里可以看到有三种取舍方式。

① 取 CA，放弃 P：放弃了分区容错性，避免分区（Partition）问题发生，可以将所有的东西（与事务相关的）都放到一台机器上，MySQL、Postgres 等数据库采用这种模式。

② 取 CP，放弃 A：放弃了高可用性，追求一致性与分区容错性。数据分布在多个网络分区的节点上，并保证这些数据的一致性，但是对于可用性的支持方面有问题，比如当集群出现问题时，节点有可能因无法确保数据是一致性的而拒绝提供服务，Google 的 BigTable、MongoDB 采用这种模式。

③ 取 AP，放弃 C：放弃了一致性，用最终一致性来确保可用性和分区容错性。这也是大部分 NoSQL 数据库使用的模式。

2. 最终一致性

在理解最终一致性之前我们先了解一致性的基本问题、一致性的分类。

在分布式系统中，为了保证高可靠性，一份数据会在多个节点中复制多份，这样就带来一致性问题了，一致性可以简单地理解为分布式系统中，在同一时间多节点的数据是一致的。而一致性又可以分为强一致性和弱一致性。强一致性可以理解为任意时刻所有节点的数据都保持严格一致，而弱一致性不要求任意时刻，弱一致性有很多不同的实现，目前分布式系统中广泛实现的是最终一致性。最终一致性的含义：过程松，结果紧，最终结果必须保持一致性。

对于一致性，可以分成从客户端和服务端两个不同的视角。从客户端来看，一致性主要指的是多进程并发访问时更新过的数据如何获取的问题。从服务端来看，则是更新如何复制分布到整个系统，以保证数据最终一致。一致性是存在并发读写时才有的问题，因此在理解一致性的问题时，要注意结合考虑并发读写的场景。

从客户端角度，多进程并发访问时，更新过的数据在不同进程如何获取的不同策略，决定了不同的一致性。对于关系型数据库，要求更新过的数据能被后续的访问都能看到，这是强一致性。如果能容忍后续的部分或者全部访问不到，则是弱一致性。如果经过一段时间后要求能访问到更新后的数据，则是最终一致性。

最终一致性根据更新数据后各进程访问到数据的时间和方式的不同，又可以区分为如下几类。

① 因果一致性：如果进程 A 通知进程 B 它已更新了一个数据项，那么进程 B 的后续访问将返回更新后的值。与进程 A 无因果关系的进程 C 的访问遵守一般的最终一致性规则。

② “读己之所写”（read-your-writes）一致性：当进程 A 自己更新一个数据项之后，它总是访问到更新过的值，绝不会看到旧值。这是因果一致性模型的一个特例。

③ 会话（Session）一致性：这是上一个模型的实用版本，它把访问存储系统的进程放到会话的上下文中。只要会话还存在，系统就保证“读己之所写”一致性。如果由于某些失败情形令会话终止，就要建立新的会话，而且系统保证不会延续到新的会话。

④ 单调（Monotonic）读一致性：如果进程已经看到过数据对象的某个值，那么任何后续访问都不会返回那个值之前的值。

单调写一致性：系统保证来自同一个进程的写操作顺序执行。要是系统不能保证这种程度的一致性，就非常难以编程了。

上述最终一致性的不同方式可以进行组合，例如单调读一致性和读己之所写一致性就可以组合实现。并且从实践的角度来看，这两者的组合，读取自己更新的数据，和一旦读取到最新的版本就不会再读取旧版本，对于此架构上的程序开发来说，会减少很多额外的烦恼。

① N —— 数据复制的份数。

② W —— 更新数据时需要保证写完成的节点数。

③ R —— 读取数据时需要读取的节点数。

如果 $W+R>N$ ，写的节点和读的节点重叠，则是强一致性。例如对于典型的一主一备同步复制的关系型数据库， $N=2$ ， $W=2$ ， $R=1$ ，则不管读的是主库还是备库的数据，都是一致的。如果 $W+R\leq N$ ，则是弱一致性。例如对于一主一备异步复制的关系型数据库， $N=2$ ， $W=1$ ， $R=1$ ，则如果读的是备库，就可能无法读取主库已经更新过的数据，所以是弱一致性。对于分布式系统，为了保证高可用性，一般设置 $N\geq 3$ 。不同的 N 、 W 、 R 组合，是在可用性和一致性之间取一个平衡，以适应不同的应用场景。如果 $N=W$ ， $R=1$ ，任何一个写节点失效，都会导致写失败，因此可用性会降低，但是由于数据分布的 N 个节点是同步写入的，因此可以保证强一致性。如果 $N=R$ ， $W=1$ ，只需要一个节点写入成功即可，写性能和可用

性都比较高。但是，读取其他节点的进程可能不能获取更新后的数据，因此是弱一致性。这种情况下，如果 $W < (N+1)/2$ ，并且写入的节点不重叠的话，则会存在写冲突。

9.2.3 分布式模型

横向扩展是 NoSQL 的一大特性，将数据库运行在服务器集群上，并且可以实现横向扩展。这种方式经济实用，其中数据分布有两条路径——复制 (Replication) 与分片 (Sharding)。数据复制属于分布式计算的范畴，它并不仅仅局限于数据库，但这里主要是指分布式数据库的复制，是将同一份数据复制至多个节点；而分片则是将不同的数据存放在不同的节点中。分片和复制是 NoSQL 里面很重要的两个概念，下面详细讲述这两点。

1. 分片

分片的基本思想就要把一个数据库切分成多个部分放到不同的数据库 (节点) 中，从而缓解单一数据库的性能问题，数据的各部分存放于不同的服务器中实现横向扩展，这种技术叫做分片。分片按照划分逻辑又可以分为水平切分和垂直切分。

(1) 水平切分

水平切分将同一个表中的不同数据拆分到不同的数据库中，相比垂直切分更复杂一些，水平切分适合表不多，但是表数据特别多的情况。有不同的实现方式，可以根据表 ID 或者 HASH 来分，图 9.3 是一个简单的分片示意图。

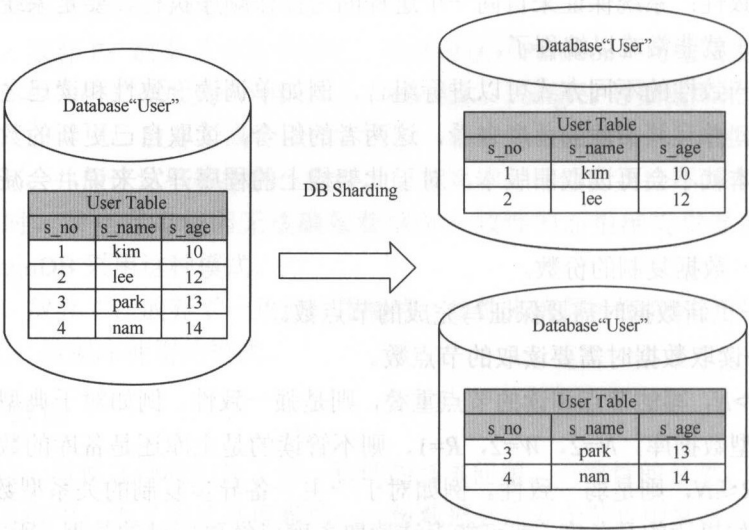


图 9.3 水平切分

Amazon 的 NoSQL 数据库 Dynamo 使用了著名的一致性 HASH 的策略。一致性哈希将整个哈希值空间组织成一个虚拟的圆环，如假设某哈希函数 H 的值空间为 $0-2^{32}-1$ (即哈希值是一个 32 位无符号整数)，各个服务器使用 HASH 进行一个哈希，具体可以选择服

务器的 IP 或主机名作为关键字进行哈希，这样每台机器就能确定其在哈希环上的位置，整个哈希空间环如图 9.4 所示，分布式系统中，一致性 HASH 是一个很重要的概念，想要了解更多信息的读者可以去查阅其他资料了解一致性 HASH 的原理以及优点。

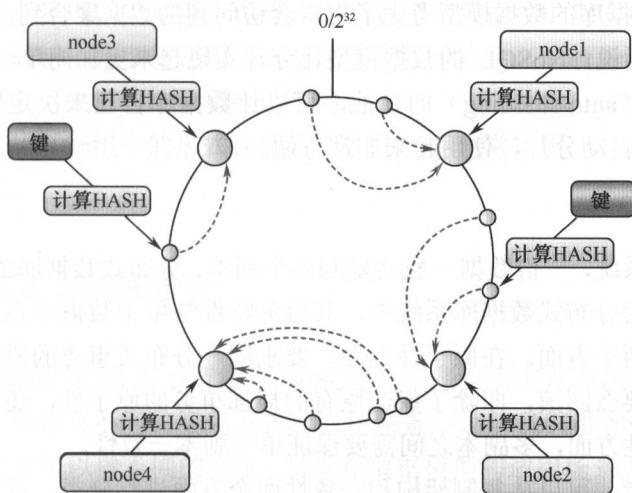


图 9.4 一致性 HASH 分布

(2) 垂直切分

理想情况下，垂直切分是最好的方案，将不同的数据分片存放于独立的节点上，每个节点只负责自身数据的读取与写入操作。规则简单，实施也更为方便，尤其适合各业务之间的耦合度非常低、相互影响很小、业务逻辑非常清晰的系统。在这种系统中，可以很容易做到将不同业务模块所使用的表拆分到不同的数据库中。根据不同的表来进行拆分，对应用程序的影响也更小，拆分规则也会比较简单清晰（图 9.5）。

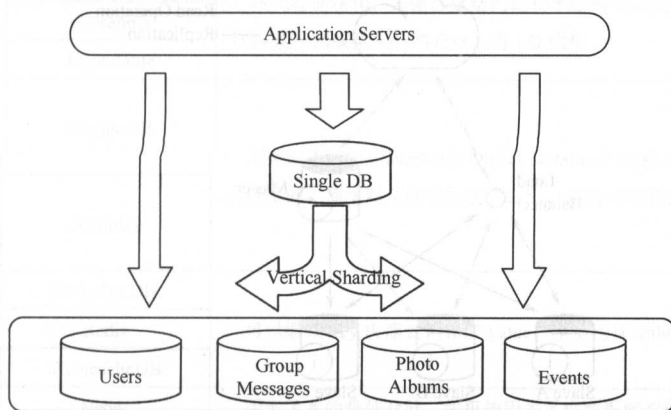


图 9.5 垂直切分

在理想的情况下，不同的服务器节点会服务于不同的用户，每位用户只需要与一台服务器通信，并且很快能获得响应。网络负载均衡地分布于各台服务器，相当于实现了负载

均衡。

理想情况比较少见，为了获得近乎理想的效果，必须保证需要同时访问的那些数据都在同一个节点上。

很多 NoSQL 数据库的数据模型考虑了将常会访问到的数据聚合到一起，所以天生适合分片的策略，换句话说，NoSQL 的数据模型让分片实现起来更加简单。另外很多 NoSQL 都提供了自动分片（auto-sharding）的功能，可以让数据库自己来决定数据的分片。例如 MongoDB 就提供了自动分片，使用起来非常方便。

2. 复制

分布式多节点系统，一份数据一般会复制多个副本，分布式数据库的复制也同样如此。

在多副本构成的分布式数据库系统中，其事务特性与单个数据库系统的差别主要表现在原子性和一致性两个方面。在原子性方面，要求同一分布式事务的所有操作在所有相关副本上要么提交，要么回滚，即除了保证原有的局部事务的原子性，还需要控制全局事务的原子性；在一致性方面，多副本之间需要保证单一副本一致性。

从功能的角度看，可以从复制架构和一致性两个方面进行分类。以复制架构分类可以分为两类：主从复制（Master-Slave）和对等复制（Peer-to-Peer）方式。

(1) 主从复制

主从复制一般由主节点 Master 接受更新请求操作，在事务操作执行完毕后，在事务提交前或后将操作通知到其他 Slave 节点。在需要频繁读取数据的情况下，主从复制最有助于提升数据访问性能。所有的读请求交给从节点来处理，以新增更多从节点的方式进行水平扩展就可以同时处理更多的数据请求。对于更新密集类型的应用，如 OLTP，则容易形成单点性能瓶颈。主从复制模式如图 9.6 所示。

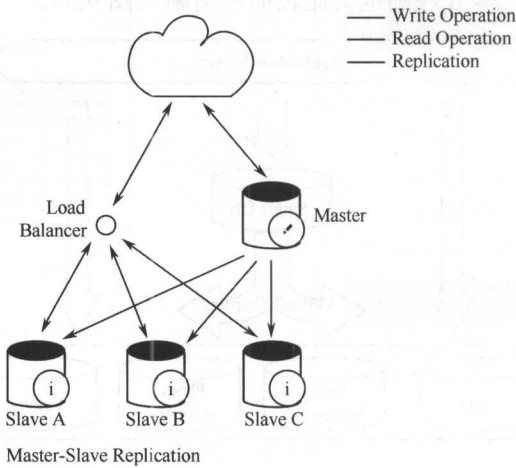


图 9.6 主从复制

(2) 对等复制

对等复制则没有中心 Master 节点的概念，节点间都是对等的，可以通过多点提高事务

吞吐率，但随之而来的是多个分布式事务之间复杂的并发控制和原子性问题。

从一致性策略来分，可以分为同步（synchronous）与异步（asynchronous），同步是在事务提交前传播更新，异步则是在提交之后才将事务操作传播到其他副本。

异步复制的优点是可以提高响应速度，但牺牲了一致性，一般实现该类协议的算法需要增加额外的补偿机制。同步复制的优点是可以保证一致性（一般通过两阶段提交协议），但是开销较大，带来了更多的冲突和死锁等问题。

异步复制和主从复制组合的形式在实际生产环境中非常实用，MySQL 的复制策略实际上就属于这种形式。

9.2.4 NoSQL 数据库分类

前面讲了很多理论知识，本小节结合 NoSQL 的一些实例来直观地理解 NoSQL。NoSQL 数据库种类繁多，有各自不同的实现方式和应用场景。之前我们讲到根据 CAP 理论，在 CAP 三个要素上取舍可以做不同的实现。另外在数据模型和适用场景上，也有不同的分类，这个网站列举了目前市面上所有的 NoSQL 数据库——<http://nosql-database.org/>。

常用的 NoSQL 大致可以分为以下四大类：面向列（Column Oriented）、面向文档（Document Oriented）、面向键值（key-value Oriented）和面向图（Graph Oriented）的数据库。

表 9.1 列出了不同分类比较典型的代表。

表 9.1 NoSQL 分类

类型	典型代表	特点
列存储	HBase	按列存储结构，方便存储结构化和半结构化数据，方便做数据压缩，针对某一系列或某几列的查询具有 I/O 优势
	Cassandra	
	Hypertable	
文档存储	MongoDB	用类似 JSON（JavaScript Object Notation）的格式存储，存储的内容是文档型的，便于对某些字段建立索引，实现关系数据库的部分功能
	CouchDB	
key-value 存储	Berkeley DB	可以通过 key 快速查询相应 value，不必考虑 value 的存储格式
	Redis	
	MemcacheDB	
图数据库	Neo4j	图形关系的最佳存储，如果使用关系型数据库存储的话，性能低，而且设计复杂
	FlockDB	

表 9.2 列出了不同存储模型的特点和性能比较，其中 key-value 存储操作简单，具有很高的性能、扩展性和灵活性；列存储灵活性相对差一些，但支持的功能相对多一些；文档

存储则可以针对某些字段建立索引，能够实现关系数据库的一些功能。

表 9.2 存储模型比较

	性 能	扩 展 性	灵 活 性	复 杂 性	功 能
关系型数据库	可变	低	低	适中	关系代数
key-value 存储	高	高	高	低	简单
列存储	高	高	适中	低	较少
文档存储	高	可变（高）	高	低	可变（低）
图数据库	可变	可变	高	高	图论

上面提到的一些 NoSQL 数据库按照数据模型和查询接口还可以细分，见表 9.3。

表 9.3 数据模型与查询接口

NoSQL 数据库	数据模型	查询 API
HBase	列族 ColumnFamily	Thrift
Cassandra	列族 ColumnFamily	Thrift, REST
MongoDB	文档	游标
CouchDB	文档	Map Reduce 视图
Riak	key/value	嵌套哈希, REST
Redis	集合 Collection	集合
Scalaris	key/value	get/put
Tokyo Cabinet	key/value	get/put
Voldemort	key/value	get/put
Neo4j	图	图

根据数据持久化的方式也可以分类，见表 9.4。

表 9.4 数据持久化设计

数据库	持久化设计
HBase	Memtable/SSTbale on HDFS
Cassandra	Memtable/SS Table
MongoDB	B-Tree
CouchDB	Append-only B-Tree
Riak	可插拔
Redis	内存数据，后台保存快照
Scalaris	内存
Tokyo Cabinet	HASH 或 B-Tree
Voldmort	可插拔（主要是 BerkeleyDB 或 MySQL）
Neo4j	存在磁盘上的链表

1. 列存储

以 HBase 为例, 主要从 HBase 的数据模型和系统架构出发, 介绍 HBase 不同于关系型数据库的存储模型, 以全面认识列存储的概念。

(1) 概要介绍

HBase 是 Hadoop Database 的缩写, 它是一个高可靠性、高性能、面向列、可伸缩的分布式存储系统, 利用 HBase 技术可在廉价 PC 服务器上搭建起大规模结构化存储集群。在前面的章节, 我们提到过 Google 公司的三大论文之一——BigTable, BigTable 也是列存储数据库, HBase 正是基于 BigTable 理论的开源实现。

HBase 主要用来存储非结构化和半结构化的松散数据, 仅能通过行键 (row key) 和行键的值域区间范围 (range) 来检索数据。

HBase 可以直接使用本地文件系统或者 Hadoop HDFS 作为数据存储基础。与 Hadoop 一样, HBase 主要依靠横向扩展, 通过不断增加廉价的商用服务器来增加计算和存储能力。HBase 的目标是处理非常庞大的表, 可以用普通的计算机处理超过 10 亿行数据并且由数百万列元素组成的数据表。HBase 中的表一般有如下特点。

- ① 大: 一个表可以有上亿行, 上百万列。
- ② 面向列: 面向列 (族) 的存储和权限控制, 列 (族) 独立检索。
- ③ 稀疏: 对于为空 (null) 的列, 并不占用存储空间, 因此, 表可以设计得非常稀疏。

从应用的角度以及流行的程度来说, HBase 是 NoSQL 中应用最广泛以及最受欢迎的数据库之一, 社区开发者多为数据库方面的资深技术专家, 有很好的发展前景。

(2) 数据模型

HBase 与其他 NoSQL 最大的不同就是它的列存储结构, 下面先介绍一下列存储以更好地理解 HBase 的数据存储模型。

早期比较影响力的 NoSQL 是 Google 公司的 BigTable, 从名字可以直观地理解它是一种带有稀疏列的无模式表结构, 类似于一个大表格的数据库, 这些采用了大表格数据模型 (BigTable-Style Data Model) 的数据库通常称为列存储 (Column Store) 数据库。

不仅如此, 列存储数据库与传统 SQL 数据库很大的区别在于数据的物理存储方式不同。大部分数据库以行为单元存储数据。然而, 有些情况下写入操作执行得很少, 但是经常需要一次读取若干行中的很多列。在这种情况下, 将某一组列作为基本数据存储单元, 效果更好。HBase 正继承了这些特点。

在 HBase 数据模型中, 有三个重要概念——行键、时间戳和列族。

- ① 行键 (Row Key): HBase 表的主键, 表中的记录按行键排序。
- ② 时间戳 (Timestamp): 每次数据操作对应的时间戳, 可以看做数据的版本号。
- ③ 列族 (Column Family): 表在水平方向由一个或者多个列族组成, 一个列族可以由任意多个列组成, 即列族支持动态扩展, 无须预先定义列的数量以及类型, 所有列均以二进制格式存储, 用户需要自行进行类型转换。

HBase 数据模型如图 9.7 所示。

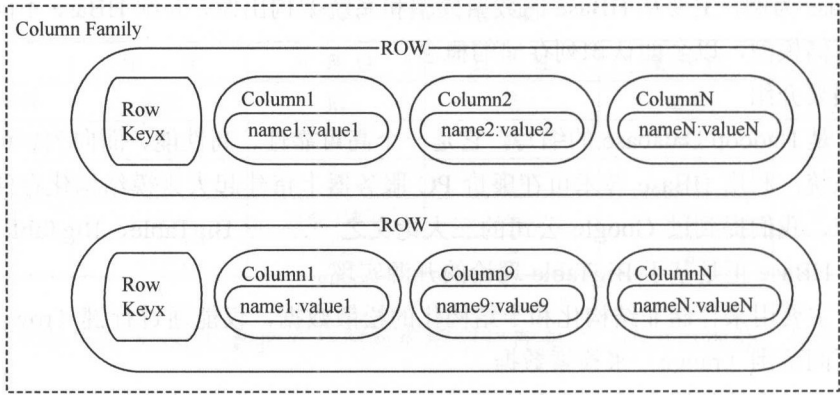


图 9.7 HBase 数据模型

表 9.5 是一个存储 Web 网页的示例。行名是一个反向 URL（即 com.taobao）。content 列族存放网页内容，anchor 列族存放引用该网页的锚链接文本。taobao 被 weibo 和 jd 引用。每个锚链接只有一个版本，而 content 列则有三个版本。

表 9.5 存储 Web 网页的示例

行键 Role Key	时间戳 Timestamp	列族 "content"	列族 "anchor"		列族 "mime"
com.taobao	T6	"<html>"	anchor:weibo.com	"weibo"	"text/html"
	T5	"<html>"	anchor:jd.com	"jd"	"text/html"
	T3				
	T2				

实际的物理存储是以行键、时间戳和单个列族的形式存储。

(3) HBase 实现与架构

HBase 也是 Master-Slave 中心化的架构，主要由 HMaster 和 HRegionServer 组成。HBase 中可以启动多个 HMaster，没有单点故障的问题，通过 ZooKeeper 的 Master 选举机制保证总有一个 Master 在运行。系统架构图如图 9.8 所示。

其中 HMaster 主要负责 Table 和 Region 的管理工作，它的主要工作如下：

- ① 管理用户对表的增删改查操作；
- ② 管理 HRegionServer 的负载均衡，调整 Region 分布；
- ③ Region 分区后，负责新 Region 的分布；
- ④ 在 HRegionServer 停机后，负责失效 HRegionServer 上 Region 的迁移。

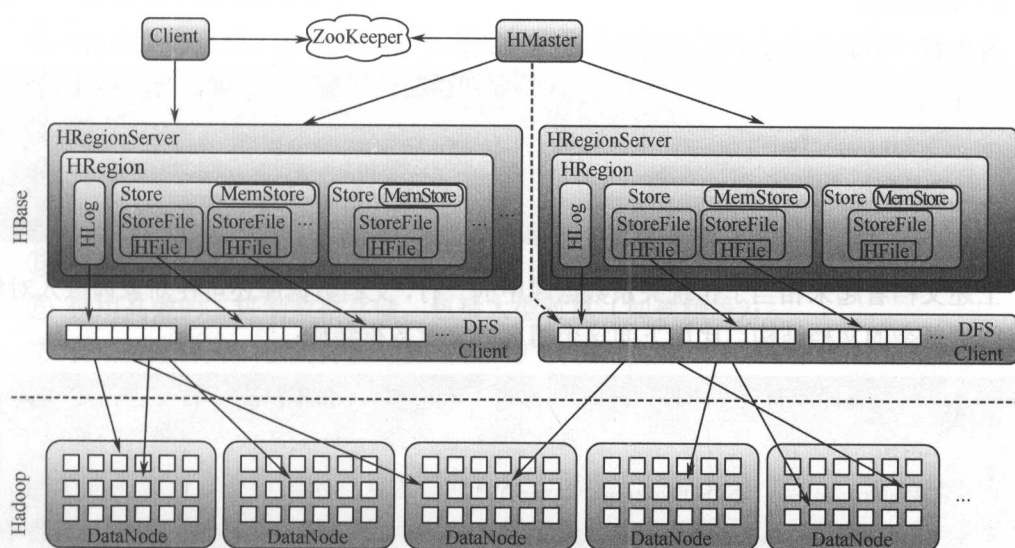


图 9.8 HBase 系统架构图

HRegionServer 是 HBase 中最核心的模块，主要负责响应用户 I/O 请求，向 HDFS 文件系统中读写数据。

HRegionServer 管理一些列 HRegion 对象，每个 HRegion 对应 Table 中一个 Region，HRegion 由多个 HStore 组成，每个 HStore 对应 Table 中一个列族的存储。

列族就是一个集中的存储单元，故将具有相同 IO 特性的列放在一个列族会更高效，依附于 Hadoop 平台，使得 HBase 可以冗余地存储海量的数据，这是 HBase 很大的一个优势。

(4) 列族数据库的应用

HBase、Cassandra 是目前最热门的两个列族数据库。

依托于强大的 Hadoop 平台，HBase 得到了快速的发展。目前 HBase 是 Apache 顶级项目，有着众多的开发人员和兴旺的用户社区。它成为一个核心的基础架构部件，运行在世界上许多大型公司（如淘宝、小米、Facebook、Twitter 等）的大规模生产环境中。

Cassandra 在 Twitter 和 Facebook 都有很多应用。

2. 文档存储

面向文档数据库用于存储、检索和管理面向文档和半结构化的数据。文档（Document）是文档数据库中的主要概念，存储的数据模型主要是文档类型，格式可以是 XML、JSON、BSON 等。这些文档具备自述性（Self-describing），呈分层的树状数据结构，可以包含映射表、集合和值。通常通过 HTTP 提供其数据访问服务，将其数据存储为 JSON 文档，并以多种语言提供 API。流行的文档数据库有 MongoDB、CouchDB、Terrastore、RavenDB 和 OrientDB 等。

1) 数据模型

首先看一个典型的文档数据结构，如下所示：

```
{
  "username": "yao",
  "scores": [ 75, 99, 87.2 ]
}
```

上述文档看起来相当于传统关系数据库中的一行，文档数据库还可在对象内嵌入对象，以获得更复杂的文档结构，比如下面这个博客的文章数据结构：

```
{
  "username": "fu",
  "BlogPostTitle": "LINQ Queries and RavenDB",
  "Date": "\Date(1266953391687+0200)\",
  "Content": "Querying RavenDB is very familiar for .NET developers who are already using LINQ for
other purposes",
  "Comments": [
    {
      "CommentorName": "Julie",
      "Date": "\Date(1266952919510+0200)\",
      "Text": "Thanks for using something I already know how to work with!",
      "UserId": "users/203907"
    },
  ]
}
```

各文档的数据模式（Schema）不同，但它们仍能放在同一个集合内，不像关系数据库那样，表中每行的数据模式都要相同。Comments 可以看成以一个子对象的形式嵌入主文档中，可以提升访问效率。模式自由（Schema-free）是文档数据库与传统关系型数据库很大的一个不同点。

文档数据库很多，CouchDB 和 RavenDB 均采用 JSON 格式存储数据，MongoDB 使用 BSON 格式存储数据。

2) MongoDB

MongoDB 是一个介于关系数据库和非关系数据库之间的产品，是非关系数据库当中功能最丰富、最像关系数据库的。其支持的数据结构非常松散，是类似 JSON 的 BSON 格式，因此可以存储比较复杂的数据类型。MongoDB 最大的特点是其支持的查询语言非常强大，其语法有点类似于面向对象的查询语言，几乎可以实现类似关系数据库单表查询的绝大部

分功能，而且还支持对数据建立索引。

它的特点是高性能、易部署、易使用，存储数据非常方便。主要功能特性有：

- ① 面向集合存储，易存储对象类型的数据；
- ② 模式自由；
- ③ 支持动态查询；
- ④ 支持完全索引，包含内部对象；
- ⑤ 支持复制和故障恢复；
- ⑥ 使用高效的二进制数据存储，包括大型对象（如视频等）；
- ⑦ 自动处理碎片，以支持云计算层次的扩展性；
- ⑧ 多语言支持，支持 Ruby、Python、Java、C++、PHP 等多种语言；
- ⑨ 文件存储格式为 BSON（JSON 的扩展形式）。

所谓面向集合，是指数据被分组存储在数据集中，被称为一个集合。每个集合在数据库中都有一个唯一的标识名，并且可以包含无限数目的文档。集合的概念类似于关系型数据库（RDBMS）里的表（Table），不同的是它不需要定义任何模式（Schema）。

模式自由意味着对于存储在 MongoDB 数据库中的文件，我们不需要知道它的任何结构定义。如果需要的话，完全可以把不同结构的文件存储在同一个数据库里。

存储在集合中的文档，被存储为键值对的形式。键用于唯一标识一个文档，为字符串类型；而值则可以是各种复杂的文件类型。我们称这种存储形式为 BSON（Binary Serialized Document Format）。

（1）组件

如图 9.9 所示，MongoDB 中有两个数据库服务器的主要组成部分。一个是 Mongod 过程，它是核心数据库服务器；另一个是 Mongos 过程，有助于数据自动分片。Mongos 被认为是一个“数据库路由器”，使得 Mongod 过程的集合看起来像是一个数据库。

（2）数据库缓存

MongoDB 消除了对单独的对象缓存层的需求。由于对象在数据库中的表示与它在内存中的表示是非常相似的，导致文件系统内存缓存的查询可以很快命中。此外，MongoDB 可以扩展到任何级别，并且提供了一个对象缓存和数据库的集成，这是非常有用的，因为它没有了从缓存中检索陈旧数据的风险。此外，也为数据库管理系统的复杂查询提供了可能。

（3）数据复制

MongoDB 支持服务器之间异步数据复制，在特定的时间，只有一台活跃服务器是可写的。由于在任何时间只有一台活跃的主服务器，因此可以取得强一致性。如果最终一致性也是可以接受的，那么也可以选择向次级服务器发送 read 请求。

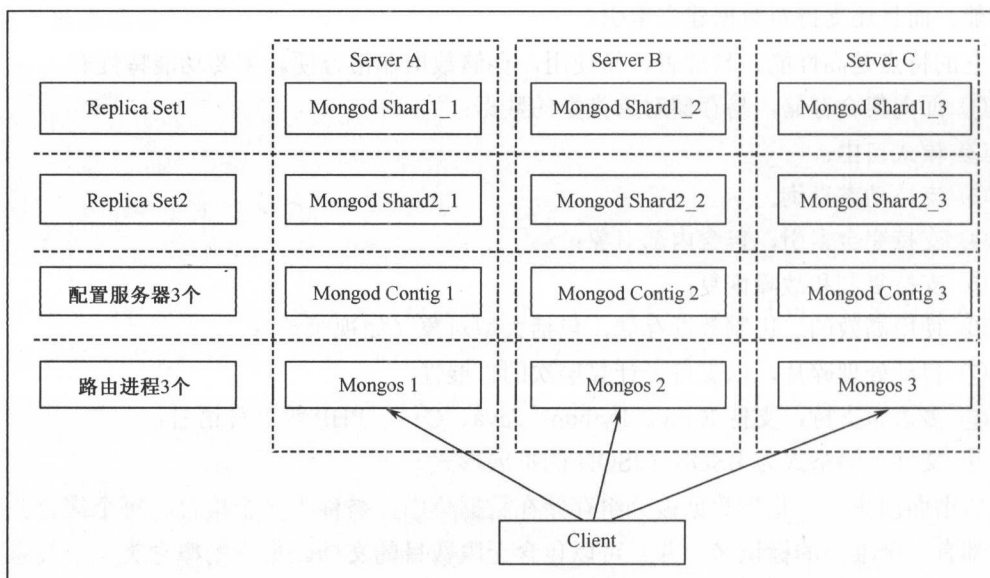


图 9.9 MongoDB 系统配置实例

(4) 分片

MongoDB 通过自动分片来进行水平扩展，分片带来了以下优点：

- ① 自动均衡负载和数据分布的变化；
- ② 轻松添加新机器；
- ③ 向外扩展至 1000 个节点；
- ④ 没有单点故障；
- ⑤ 自动故障转移。

MongoDB 也存在以下两大问题。

第一是删除锁定问题，当批量删除记录时，数据库会锁定阻止读写。这意味着进行数据清理时会让网站应用失去响应。

第二是内存占用问题，MongoDB 用了操作系统的内存文件映射，这导致操作系统会把所有空闲内存都分配给 MongoDB，而当 MongoDB 没有这个需要时，MongoDB 并不会主动释放占用的空闲内存，除非重启数据库。

3) 应用场景与实例

文档数据库的应用场景可以大致分为以下几类。

① 对象或 JSON 数据存储：文档数据库通常很好地支持 JSON 文档，而像内容管理系统与博客平台这类系统的数据多为 JSON 文档，很适合用文档数据库来存储。

② 网站数据与分析：适合实时插入、更新与查询，并具备网站实时数据存储所需的复制及高度伸缩性。

③ 大尺寸、低价值数据存储：使用传统的关系型数据库存储一些数据时可能会比较昂贵，在此之前，很多时候程序员往往会选择传统的文件进行存储。

3. key-value 存储

键值对存储是数据库最简单的组织形式。基本上所有的编程语言都带有应用在内存中的键值对存储。C++STL 的映射容器（Map Container）和 Java 的 HashMap 以及 Python 的字典类型都是键值对存储。这些容器大部分底层实现都使用哈希表或者某种平衡树（例如 B-树或者红黑树）。有时候数据太大而装不进内存，或者需要数据持久化等。在这些情况下，就必须使用文件系统。NoSQL 数据库是一个很好的替代品，下面我们了解一下 NoSQL 数据库。

1) 简介

直观来看，key-value 数据库可能是最简单的 NoSQL 数据库。客户端可以根据键来查询值，值只是数据库存储的数据而已，它不关心也无须知道其中的内容，由于键值数据库总是通过主键来访问，因此它们具有极高的并发读写性能，且易于扩展。图 9.10 显示了一个典型的 key-value 实例。

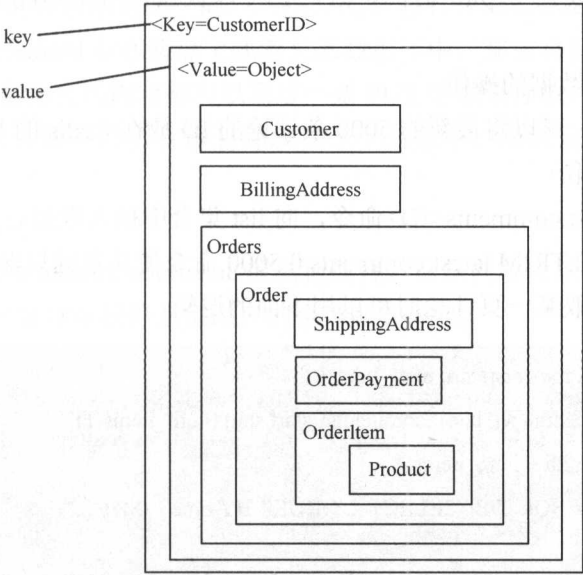


图 9.10 key-value 实例

key 是客户 id，value 是一个对象，对象里面可能也有嵌套关系。当然不同的 NoSQL 对 value 类型的支持是不同的。

流行的键值数据库有 Redis、Riak、Memcached、BerkeleyDB、Amazon DynamoDB 等。其中有一些是用 C、C++语言开发的，性能相当出色，除此之外它们还有自己独特的功能。例如 Redis 中值的类型不限于字符串，还支持列表、哈希、集合和有序集，并在这些数据结构类型上定义了一套强大的 API。

2) Redis

Redis 是一个开源的基于内存、键值对的数据库，使用 C 语言编写。从 2013 年 5 月开始，Redis 的开发由 Pivotal 赞助。之前，其开发由 VMware 赞助。根据月度排行网站 DB-Engines.com 的数据显示，Redis 是最流行的键值对存储数据库。

Redis 的高性能在很大程度上弥补了 Memcached 的不足，在部分场合可以对关系数据库起到很好的补充作用。它跟 Memcached 类似，不过数据可以持久化，而且支持的数据类型很丰富，有字符串、链表、集合和有序集合。它支持在服务器端计算集合的并、交和补集等，还支持多种排序功能。所以 Redis 也可以被看成一个数据结构服务器。

Redis 的持久化模式可以分为半持久化模式和全持久化模式。Redis 的所有数据都保存在内存中，会不定期地通过异步方式保存到磁盘上，称为半持久化模式；也可以把每一次数据变化都写入一个 append only file 里面，称为全持久化模式。

Redis 可以用来实现很多有用的功能，比如用它的 list 来做 FIFO 双向链表，实现一个轻量级的高性能消息队列服务，用它的 set 可以做高性能的 tag 系统等。另外 Redis 也可以对存入的 key-value 设置 expire 时间，因此也可以被当作一个功能加强版的 Memcached 使用。

(1) 取最新 N 个数据的操作

通过下面的方式，可以将最新的 5000 条评论的 ID 放在 Redis 的 list 集合中，并将超出集合部分从数据库获取

使用 LPUSH latest.comments<ID>命令，向 list 集合中插入数据。

插入完成后再用 LTRIM latest.comments 0 5000 命令使其永远只保存最近 5000 个 ID。

然后在客户端获取某一页评论时可以用下面的逻辑：

```
func get_latest_comments(start, num_items):
    id_list = redis.lrange("latest.comments", start, start+num_items-1)
    IF id_list.length < num_items
        id_list = SQL_DB("SELECT ... ORDER BY time LIMIT ...")
    END
    RETURN id_list
END
```

如果还有不同的筛选维度，比如某个分类的最新 N 条，那么可以再建一个按此分类的 list，在只存 ID 的情况下，Redis 是非常高效的。

(2) 排行榜应用，取 TOP N 操作

这个应用以某个条件为权重，需要用到 sorted set，将要排序的值设置成 sorted set 的 score，将具体的数据设置成相应的 value，每次只需要执行一条 ZADD 命令即可。

(3) 需要精准设定过期时间的应用

将上面提到的 sorted set 的 score 值设置成过期时间的时间戳，就可以简单地通过过期

时间排序, 定时清除过期数据, 不仅可以清除 Redis 中的过期数据, 还可以把 Redis 里的这个过期时间当成对数据库中数据的索引, 用 Redis 找出哪些数据需要过期删除, 然后精准地从数据库中删除相应的记录。

(4) 计数器应用

Redis 的命令都是原子性的, 可以轻松地利用 INCR、DECR 命令来构建计数器系统。

(5) 实时系统和反垃圾系统

通过上面提到的 set 功能, 可以知道一个终端用户是否进行了某项操作, 可以找到其操作的集合并进行分析、统计、对比等。

(6) 构建队列系统

使用 list 可以构建队列系统, 使用 sorted set 甚至可以构建有优先级的队列系统。

3) 应用场景

key-value 数据适用的场景如下。

(1) 存放会话信息

通常来说, 每一次会话都是唯一的, 所以分配给它们的 sessionid 值也各不相同。如果应用程序原来要把 sessionid 存在磁盘上或者关系数据库中, 那么将其迁移到键值数据库之后会受益良多, 因为全部会话内容都可以通过一条 PUT 请求来存放, 而且只需要一条 GET 请求就能取得。由于会话中的所有信息都放在一个对象中, 所以这种“单请求操作”很迅速。

(2) 用户配置信息

几乎每位用户都有 userid、username 或其他属性, 这些内容可以全部放在一个对象中, 以便以一次 GET 操作就可以获得某位用户的全部配置信息。

(3) 购物车数据

电子商务网站的用户都会与其购物车相绑定。由于购物车里面的内容要在不同时间、不同浏览器、不同电脑、不同会话中保持一致, 所以可把购物信息放在 value 属性中, 并将其绑定在 userid 这个键上。

4. 图数据库

1) 概念

图数据库可以简单地理解为用图数据结构来存储数据, 每个对象是一个节点, 对象之间的关系是一条边。相对于关系数据库来说, 图数据库善于处理大量复杂、互连接、低结构化的数据, 这些数据变化迅速, 需要频繁查询。在关系数据库中, 由于这些查询会导致大量的表连接, 从而导致性能问题, 而且在设计使用上也不方便。图数据库适用于社交网络、推荐系统等专注于构建关系图谱的系统。

如图 9.11 所示, 在图数据库中, 对象即为用户 (user), 对象之间会有一系列的关系, 错综复杂。

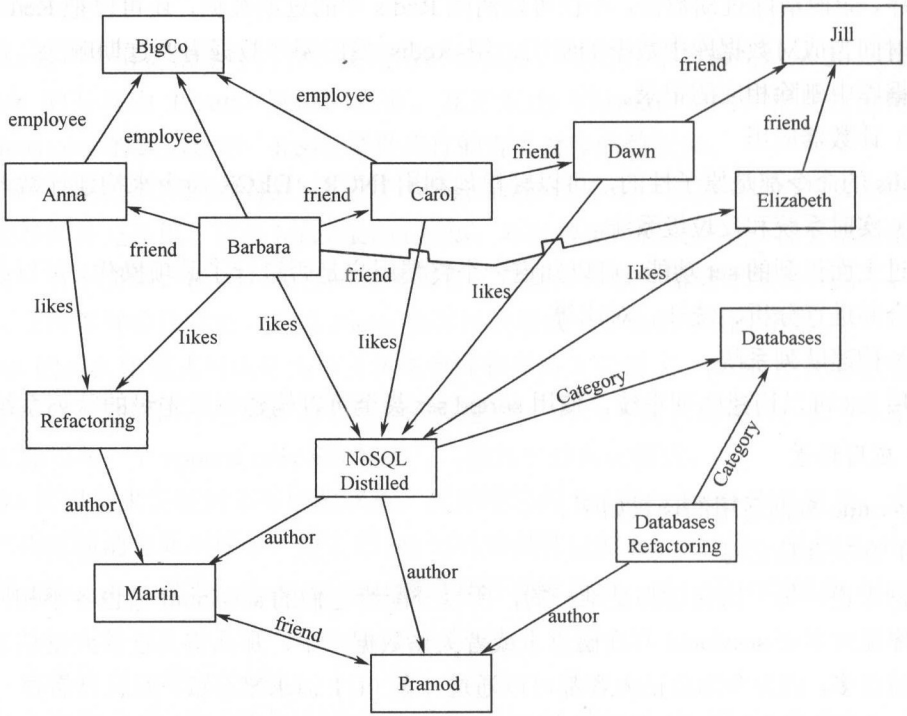


图 9.11 图数据库示例

2) Neo4j

对于很多应用来说，其中的领域对象模型本身就是一个图结构。对于这样的应用，使用 Neo4j 这样的图数据库进行存储是最合适的，因为在进行模型转换时的代价最小。以基于社交网络的应用为例，用户作为应用中的实体，通过不同的关系关联在一起，如亲人关系、朋友关系和同事关系等。不同的关系有不同的属性。比如，同事关系所包含的属性包括所在公司的名称、开始的时间和结束的时间等。对于这样的应用，使用 Neo4j 来进行数据存储，不仅实现起来简单，后期的维护成本也比较低。

Neo4j 使用“图”这种最通用的数据结构来对数据进行建模，使得 Neo4j 的数据模型在表达能力上非常强。链表、树和散列表等数据结构都可以用图来表示。Neo4j 同时具有一般数据库的基本特性，包括事务支持、高可用性和高性能等。Neo4j 已经在很多生产环境中得到了应用。流行的云应用开发平台 Heroku 也提供了 Neo4j 作为可选的扩展。

(1) 节点和关系

Neo4j 中最基本的概念是节点 (Node) 和关系 (Relationship)。节点表示实体，由 `org.neo4j.graphdb.Node` 接口来表示。在两个节点之间，可以有不同的关系。关系由 `org.neo4j.graphdb.Relationship` 接口来表示。每个关系由起始节点、终止节点和类型三个要素组成。起始节点和终止节点的存在，说明了关系是有方向的，类似于有向图中的边。不过在某些情况下，关系的方向可能并没有意义，会在处理时被忽略。所有的关系都是有类

型的，用来区分节点之间意义不同的关系。在创建关系时，需要指定其类型。关系的类型由 `org.neo4j.graphdb.RelationshipType` 接口来表示。节点和关系都可以有自己的属性。每个属性是一个简单的名值对。属性的名称是 `String` 类型的，而属性的值则只能是基本类型、`String` 类型以及基本类型和 `String` 类型的数组。一个节点或关系可以包含任意多个属性。对属性进行操作的方法声明在接口 `org.neo4j.graphdb.PropertyContainer` 中。`Node` 和 `Relationship` 接口都继承自 `PropertyContainer` 接口。`PropertyContainer` 接口中常用的方法包括获取和设置属性值的 `getProperty` 和 `setProperty`。下面通过具体的示例来说明节点和关系的使用。

(2) 编程实例

① 可以用 `enum` 来创建关系类型，如创建一个关系类型“KNOWS”，如下所示：

```
private static enum RelTypes implements RelationshipType
{
    KNOWS
}
```

② 启动数据库服务器，如果给定的 `Path` 不存在则会重新创建一个。

```
graphDb = new GraphDatabaseFactory().newEmbeddedDatabase( DB_PATH );
registerShutdownHook( graphDb );
```

③ 执行一些操作（Operation），Neo4j 里的操作都被封装在事务（Transaction）里面。如图 9.12 所示，创建一个简单的 Hello World 节点关系。

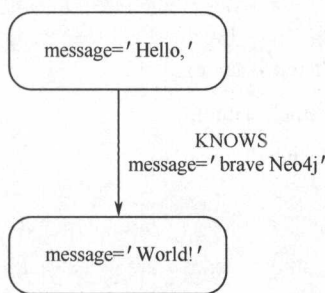


图 9.12 Hello World 节点关系

代码如下所示：

```
try ( Transaction tx = graphDb.beginTx() )
{
    firstNode = graphDb.createNode();
    firstNode.setProperty( "message", "Hello, " );
    secondNode = graphDb.createNode();
```

```

secondNode.setProperty( "message", "World!" );
relationship = firstNode.createRelationshipTo( secondNode, RelTypes.KNOWS );
relationship.setProperty( "message", "brave Neo4j " );
tx.success();
}

```

④ 打印输出结果，会输出 “Hello, brave Neo4j World!”。

```

System.out.print( firstNode.getProperty( "message" ) );
System.out.print( relationship.getProperty( "message" ) );
System.out.print( secondNode.getProperty( "message" ) );

```

⑤ 使用索引。

当Neo4j数据库中包含的节点比较多时，要快速查找满足条件的节点会比较困难。Neo4j提供了对节点进行索引的能力，可以根据索引值快速地找到相应的节点。下面的示例是索引的一个基本用法。

```

public void useIndex() {
    GraphDatabaseService db = new EmbeddedGraphDatabase("music");
    Index<Node> index = db.index().forNodes("nodes");
    Transaction tx = db.beginTx();
    try {
        Node node1 = db.createNode();
        String name = "user 1";
        node1.setProperty("name", name);
        index.add(node1, "name", name);
        node1.setProperty("gender", "man");
        tx.success();
    } finally {
        tx.finish();
    }
    Object result = index.get("name", "usser 1").getSingle()
        .getProperty("gender");
    System.out.println(result); // 输出为“man”
}

```

通过 GraphDatabaseService 接口的 index 方法可以得到管理索引的 org.neo4j.graphdb.index.IndexManager 接口的实现对象。Neo4j 支持对节点和关系进行索引。通过 IndexManager 接口的 forNodes 和 forRelationships 方法可以分别得到节点和关系上的索

引。索引通过 `org.neo4j.graphdb.index.Index` 接口来表示，其中的 `add` 方法用来把节点或关系添加到索引中，`get` 方法用来根据给定值在索引中进行查找。

3) 适用场景

根据图数据库的特性，分析图数据库的应用场景。图数据库大致有以下三种应用场景。

(1) 互连的数据

部署并使用图数据库来处理社交网络非常高效，社交图里人与人的关系是多样的，不是只有好友关系。任何富含多链接关系的领域都适合用图数据库来处理。

(2) 设计线路和基于位置的服务

物流投递过程中的每个地点或者地址都是一个节点，可以把送货员投递货物时所经全部节点建模为一张节点图。节点之间的关系可带有距离属性，以便高效投递货物。

(3) 推荐系统

在系统中创建节点与关系时，可以用它们为客户推荐信息。

9.3 NewSQL

NoSQL 数据库可提供良好的扩展性和灵活性，但它们也有自己的不足。由于不使用 SQL，NoSQL 数据库系统不具备高度结构化查询等特性，另外它对事务的支持不是很完善。另外不同的 NoSQL 数据库都有自己的查询语言，这使得很难规范应用程序接口。NewSQL 是对各种新的可扩展/高性能数据库的简称，这类数据库不仅具有 NoSQL 对海量数据的存储管理能力，还保持了传统数据库支持 ACID 和 SQL 等特性。

NewSQL 结合了传统关系型数据库和灵活的 NoSQL 数据库的优点，可以预测 NewSQL 是未来数据库的发展方向。

9.3.1 系统分类

NewSQL 系统虽然内部结构变化很大，但是它们有两个显著的共同特点：它们都支持关系数据模型，它们都使用 SQL 作为其主要的接口。已知的第一个 NewSQL 系统叫做 H-Store，它是一个分布式并行内存数据库系统。目前 NewSQL 系统大致分为以下三类。

1. 新的架构

第一种 NewSQL 系统是全新的数据库平台，它们均采取了不同的设计方法。它们大致分为以下两类。

① 工作在一个分布式集群的节点上，其中每个节点拥有一个数据子集。SQL 查询被分成查询片段发送给自己所在的数据的节点上执行。这些数据库可以通过添加额外的节点来线性扩展。这一类典型的数据库有 Google Spanner、VoltDB、Clustrix 和 NuoDB。

② 通常有一个单一的主节点的数据源。它们有一组节点用来做事务处理，这些节点接到特定的 SQL 查询后，会把它所需的所有数据从主节点上取回来后执行 SQL 查询，再返回结果。

2. SQL 引擎

第二种是高度优化的 SQL 存储引擎。这些系统提供了统一的 MySQL 编程接口，扩展性比 InnoDB 这样的内置引擎更好。这类系统的典型代表有 TokuDB 和 MemSQL。

3. 透明分片

这类系统提供了分片的中间件层，这样数据库可以跨多节点分片。这类数据库包括 ScaleBase、dbShards 和 Scalearc。

9.3.2 Google Spanner

本节以比较出名的 NewSQL 数据库 Google Spanner 为例，简单介绍一下 NewSQL 的功能与实现。

1. Spanner 简介

BigTable 展示了一个简洁、优美、具有高可扩展性的分布式数据库系统，引发了 NoSQL 浪潮。然而 Spanner 的设计者们指出了 BigTable 在应用中遇到的一些阻力，具体如下：

- ① 缺少类似 SQL 的界面，缺少关系数据库拥有的丰富的功能；
- ② 只支持单行事务，缺少跨行事务；
- ③ 需要在跨数据中心的多个副本间保证一致性。

这些来自应用开发者的需求催生了 Spanner，一个既拥有 key-value 系统的高可扩展性，也拥有关系数据库的丰富功能（包括事务、一致性等特性）的系统。这类兼顾可扩展性和关系数据库功能的产品被称为 NewSQL。

Spanner 是 Google 公开的新一代分布式数据库，它既具有 NoSQL 系统的可扩展性，也具有关系数据库的功能。例如，它支持类似 SQL 的查询语言，支持表连接，支持事务（包括分布式事务）。Spanner 可以将一份数据复制到全球范围的多个数据中心，并保证数据的一致性。一套 Spanner 集群可以扩展到上百个数据中心、百万台服务器和上 T 条数据库记录的规模。目前，Google 广告业务的后台（F1）已从 MySQL 分库分表方案迁移到了 Spanner 上。

Spanner 的设计显示了 Google 多年来在分布式存储系统领域中经验的积累和沉淀，它采用了 Megastore 的数据模型、Chubby 的数据复制和一致性算法，而在数据的可扩展性上使用了 BigTable 中的技术。新颖之处在于，它使用高精度和可观测误差的本地时钟来判断分布式系统中事件的先后顺序。Spanner 代表了分布式数据库领域的新趋势——NewSQL。

2. 数据模型

传统的 RDBMS（例如 MySQL）采用关系模型，有丰富的功能，支持 SQL 查询语句。而 NoSQL 数据库多在 key-value 存储之上增加有限的功能，如列索引、范围查询等，但具有良好的可扩展性。Spanner 继承了 Megastore 的设计，数据模型介于 RDBMS 和 NoSQL 之间，提供树形、层次化的数据库 Schema，一方面支持类 SQL 的查询语言，提供表连接等关系数据库的特性，功能上类似于 RDBMS；另一方面，整个数据库中的所有记录都存储在同一个 key-value 大表中，实现上类似于 BigTable，具有 NoSQL 系统的可扩展性。

在 Spanner 中，应用可以在一个数据库里创建多个表，同时需要指定这些表之间的层次关系。显然所有表的主键都将根节点的主键作为前缀，Spanner 将根节点表中的一条记录和以其主键作为前缀的其他表中的所有记录的集合称为一个 Directory。Directory 是 Spanner 中对数据进行分区、复制和迁移的基本单位，应用可以指定一个 Directory 有多少个副本，分别存放在哪些机房中，例如把用户的 Directory 存放在这个用户所在地区附近的几个机房中。

这样的数据模型有以下几个好处。

① 一个 Directory 中所有记录的主键都具有相同前缀，在存储到底层 key-value 大表时，会被分配到相邻的位置。如果数据量不是非常大，会位于同一个节点上，这不仅提高了数据访问的局部性，也保证了在一个 Directory 中发生的事务都是单机的。

② Directory 还实现了从细粒度上对数据进行分区。整个数据库被划分为百万个甚至更多个 Directory，每个 Directory 可以定义自己的复制策略。这种 Directory-based 的数据分区方式比 MySQL 分库分表时 Table-based 的粒度要细，而比 Yahoo 的 PNUTS 系统中 Row-based 的粒度要粗。

③ Directory 提供了高效的表连接运算方式。在一个 Directory 中，多张表上的记录按主键排序，交错地存储在一起，因此进行表连接运算时无须排序即可在表间直接进行归并。

3. 复制和一致性

Spanner 使用 Paxos 协议在多个副本间同步 redo 日志，从而保证数据在多个副本上是一致的。Google 的工程师钟情于 Paxos 协议，Chubby、Megastore 和 Spanner 等一系列产品都是在 Paxos 协议的基础上实现一致性的。Paxos 协议的具体原理可以参考互联网上的文献资料。

9.3.3 MemSQL

1. MemSQL 简介

MemSQL 是一款内存数据库，它通过将数据存在内存中，将 SQL 语句预编译为 C++ 而获得极快的执行效率。MemSQL 号称是世界上最快的分布式关系型数据库，兼容 MySQL 但快 30 倍，能实现每秒 150 万次事务。MemSQL 由前 Facebook 工程师 Eric Frenkiel 和微软 SQL Server 高级工程师 Nikita Shamgunov (CTO) 联合创建，MemSQL 的高性能数据库还参照了 Facebook 的脚本，有着强烈的 Facebook 印记。2011 年 7 月，MemSQL 获得了 Ashton Kutcher、SV Angel、Paul Buchheit 以及 New Enterprise Associates 等 14 家风投的青睐，融资 210 万美元；仅一年以后又再次获得了 IA Ventures 和 Digital Sky Technologies 两家共计 300 万美元的风险投资。

MemSQL 具有以下几大特点。

- ① 高效率：MemSQL 执行效率比传统的基于磁盘的数据库要高 30 倍，它优于其他内存数据库，因为它将 SQL 语句预编译为 C++。
- ② 强大的 SQL 执行能力：支持全功能的关系型数据库，开发者不必修改现有程序即可获得 NoSQL 键值存储系统的效率。
- ③ 横向和纵向扩展：MemSQL 支持纵向扩展，CPU 越好效率就越高，而且支持向多 CPU 扩展；此外，MemSQL 还可与 MySQL 节点结合起来处理 PB 级的负载。
- ④ 默认支持数据持久性：MemSQL 默认支持数据从内存到磁盘/SSD 的同步，保证数据的安全可靠。
- ⑤ 安装简便：只需 30 秒即可完成安装并使用 MemSQL，兼容 MySQL，学习曲线平滑。
- ⑥ 支持 JSON 格式的数据处理。

2. MemSQL 架构

MemSQL 是 ShareNothing 的分布式架构，如图 9.13 所示。

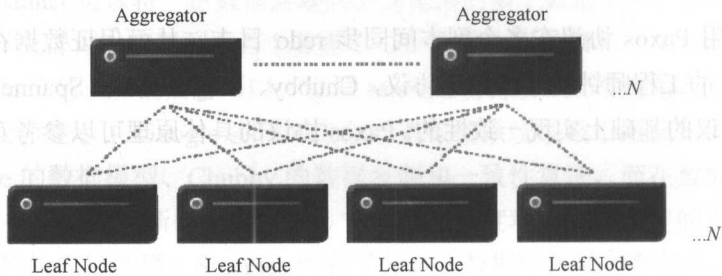


图 9.13 MemSQL 架构

MemSQL 架构主要分为两层：Aggregator 和 Leaf。Aggregator 存储元数据，负责分发 sql 给 Leaf，然后综合 Leaf 的查询结果。Aggregator 可以有多个，其中有一个为主 Aggregator。主 Aggregator 还可以执行 DDL 和负责 Leaf 的 auto-failover。其他普通 Aggregator 则不行。当主 Aggregator 失效时，可以从其他普通 Aggregator 中选择一个通过 sql 命令设置为主 Aggregator。

Leaf 上存储真正的数据，数据通过主键 hash 存储到各个 Leaf 节点，Leaf 之间的数据均匀分布，不会倾斜。目前还不支持范围分区。在建库时指定分区数，分区数应该为 Leaf 节点数的整数倍，一般设为 8 倍。每个 Leaf 都是一个分库，分取数据存储在分库的表中。表支持二级索引，同时支持以主键为前缀的唯一索引。

MemSQL 有两种表，即参照表（reference table）与分布表（sharded table）。

参照表：数据分布在主 Aggregator 和每个 Leaf 节点。每个节点的数据都是完整的（没有分区）。参照表通过复制从主 Aggregator 向每个 Leaf 节点同步数据。另外参照表的写操作只能在主 Aggregator 进行。

分布表：数据通过 hash 分片存储在每个 Leaf 节点，每个 Leaf 节点只有部分数据。

3. MemSQL 数据冗余策略

MemSQL 有可用组（Availability Group），每个组是一些 Leaf 的集合。组与组之间是冗余存储的。目前最多支持两个组。以两个组为例，每个组都包含完整数据，每个分区表在两个组都有一份副本。

4. MemSQL 可扩展性

支持动态增删 Leaf 节点，但需要执行 rebalance partitions 命令来重新分布分区数据。rebalance 操作都是在线进行的，即操作过程中不影响数据正常访问。rebalance 操作的单位是库，最小粒度是 partition。

Leaf 节点状态变化如图 9.14 所示。

Leaf 节点有 4 种状态，分别是 Unknow、Online、Offline 和 Detached。状态间会相互转换。下面来解释一下这 4 个状态的具体含义。

① Unknow：在这个状态下，Leaf 还没有加入集群。这是在执行 ADD LEAF 命令把 Leaf 加入系统之前的状态。Unknow 状态下的 Leaf 在执行 SHOW LEAVES 时不会被显示。如果执行 REMOVE LEAF 命令，Leaf 会转为 Unknown 状态。

② Online：这是 Leaf 默认的健康状态。在 Online 状态下，Leaf 是分布式系统中活动的一员，随之等待任务来临。

③ Offline：如果 Master Aggregator 不可达，则 Leaf 为 Offline 状态，Master Aggregator 会向每个 Leaf 节点发送心跳信息以检查离线的节点。

④ Detached：一旦 Leaf 节点又重新可达，节点将从 Offline 状态转换到 Detached 状态。可以通过 ATTACH LEAF 命令重新将 Leaf 节点激活。这个命令将自动重用当前 Leaf 节点上的数据，这可以减少冗余。

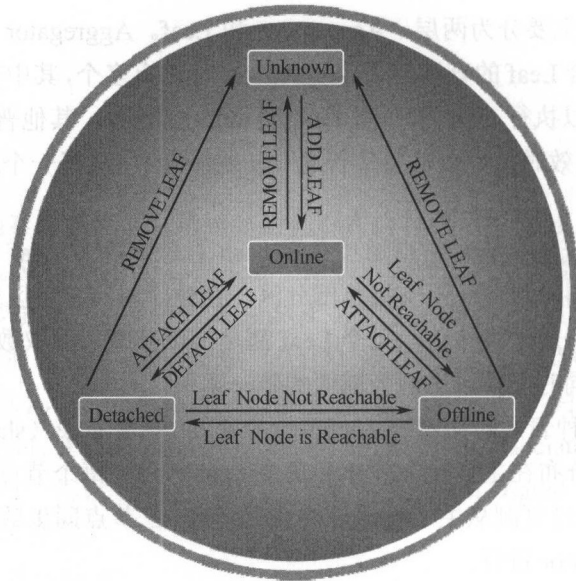


图 9.14 Leaf 节点状态变化

9.3.4 VoltDB

1. VoltDB 简介

VoltDB 是一个内存中的开源 OLTP SQL 数据库，能够保证事务的完整性（ACID）。它结合了 NoSQL 和传统关系型数据库的优点，提供了 NoSQL 数据库的可伸缩性和传统关系数据库系统的 ACID 一致性，它是 Postgres 和 Ingres 联合创始人 Mike Stonebraker 领导开发的下一代开源数据库管理系统。它能在现有的廉价服务器集群上实现每秒数百万次数据处理。

为了最大化吞吐量，数据保存在内存中（而不是在硬盘中），这样可以有效消除缓冲区管理。VoltDB 通过 SQL 引擎把数据分发给集群服务器的每个 CPU 进行处理。每个单线程分区自主执行，消除锁定和阻塞的需求。VoltDB 可以通过简单的在集群中增加附加节点的方式实现性能的线性增加。

从硬件上看，VoltDB 基于 PC+以太网+本地存储；从体系结构上看，其内部是一个 ShareNothing 的内存数据库，通过并行单线程来保证事务一致性和高性能，所有事务被实现为 Java 存储过程，所有存储过程（事务）均全局有序，由于避免了锁的使用，因此可以保证每个事务在所有分区上并行执行完成后才继续执行下一个事务，事务不会乱序执行。存储过程内部支持分组、多路 Join、聚合、函数等，使用单事务多 SQL 可以有效提高吞吐量。VoltDB 的可靠性通过冗余和自动恢复来保证。

VoltDB 值得关注的特性是自动数据分区，数据表会被自动分配到集群节点。另外一个特性是自动快照，这样在一个事务内部无须进行 IO 操作，可以在微秒级别完成事务，据说性能可提高 50 倍。第三个特性是异步事务提交。从某种意义上看，VoltDB 是一个共

享内存的集群。VoltDB 支持多节点并行事务处理，理论上不存在节点上限，不过 VoltDB 开发人员最大测试集群是 20 个节点。

2. VoltDB 数据分布

VoltDB 使用 ShareNothing 结构，整个数据库的数据分散到集群的多台机器上。VoltDB 的数据分布策略是基于哈希的，存储在 VoltDB 中的每一张表，对数据的主键哈希取模后的结果对应于数据存储的节点。相比于 BigTable 基于主键的连续范围分段的方法，哈希方法的好处是数据分散均匀，没有动态数据调整的烦恼。但也有很多缺点，采用这种方法后，集群的规模是事先确定好的，新增机器需要停止服务后重新分布数据。另外，数据哈希被分散后，数据的连续性被打乱了，在这个数据结构上做范围查询需要动用服务这张表的所有机器（图 9.15）。

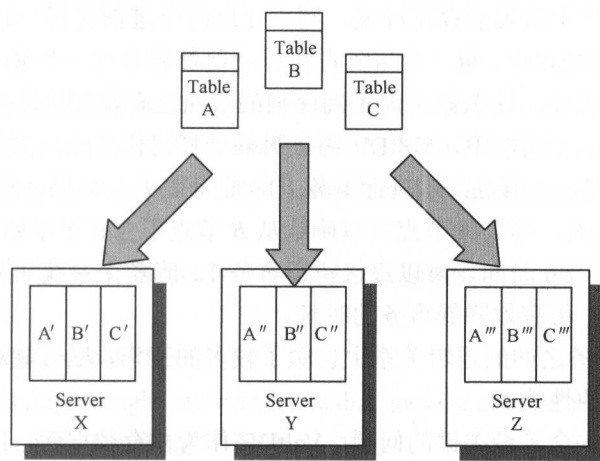


图 9.15 VoltDB 数据分布分区策略

3. VoltDB 事务

VoltDB 支持大多数 SQL 语句。其单个的 SQL 语句和存储过程都支持事务的所有 ACID 特性，是一个完全支持事务处理的系统。VoltDB 采取了一种比较极端的事务提交方式来支持复杂的事务操作，同时保证快速的执行过程。虽然 VoltDB 支持部分 SQL 语句接口，但是不允许用户使用传统的“BEGIN TRANSACTION”和“END TRANSACTION”的语法模式，而是完全基于存储过程。用户通过写存储过程完成应用程序的逻辑，作为一个先置条件将存储过程提交到 VoltDB。运行时，用户程序调用存储过程完成事务操作，所有事务的运行逻辑是由 VoltDB 在服务器进程中完成的。这种方式保证了事务不会被人为打断，并且服务器可以预先判断各个事务的逻辑，也为事务并发处理挖掘信息。

VoltDB 采取所有修改在每一个副本上单独更新的方式，来保证同一份数据在多个副本上的一致性。VoltDB 的事务并发控制需要依赖于集群内所有机器的时间是一致的。可以使用类似 NTP 的时间同步协议来保证机器之间的时间差远远小于一个交换机下的两台机器的 Round Trip 时间。VoltDB 对于用户每一次事务的调用分配一个时间戳，并且保证这个时间

戳是全局有序的，虽然时间戳是由集群中的各台机器独自分配的，但是加上机器的序号，可以保证（机器序号，时间戳）的组合值是全局有序的。一台服务器执行事务之前，需要等待 Round Trip 时间，如果其他机器没有开始比自己更早的事务，那么就执行自己的事务。以这种方式保证集群内多台机器之间事务的有序性。数据的多个副本的更新操作也都以相同的顺序进行修改，所有副本之间保证了一致性。

为了充分发挥多核机器的性能，而又不引入多线程执行事务的复杂性，VoltDB 的数据分片规模是按照集群核数来划分的。一台物理机器上可能运行多个 VoltDB 服务器进程，每个进程对应于一个核，服务器进程之间都是通过网络进行通信的。在单个进程内，只使用单线程，所有的事务执行都是顺序进行的。

多个事务在多个服务器节点同时执行，VoltDB 保证如果事务之间有冲突，那么事务的执行是完全隔离的，即达到 SERIALIZABLE ISOLATION。VoltDB 会事先分析好存储过程之间的关系，如果两个事务可能存在冲突，则不让这两个进程在同一时间执行。

在 VoltDB 的并发处理中，每一个事务在执行之前都要等待一个 Round Trip 时间，这显然会增加事务执行的时延。这么做是为了确保别的节点没有发起比这个事务更早的事务，保证事务执行的顺序。在实现中，VoltDB 用了另外一种优化方法。例如 A 和 B 两个节点，分别要执行事务 1 和 2，A 节点开始执行事务 1 的时间是 T_1 ，如果 A 收到 B 发了事务 2 的执行需求，并且 $T_2 > T_1$ ，那么 A 节点可以确认从 B 节点不会有更早的事务再发送过来，A 节点就不必等 Round Trip 时间，可以直接执行事务 1。当整个系统压力比较大时，这个优化方法效果尤其明显，可有效降低事务的时延。

VoltDB 在处理事务之间的逻辑关系上也做了较多的工作，尽可能对事务分门别类进行处理，以期获得更好的性能。

数据库中数据的安全是最关键的问题。VoltDB 作为内存数据库，用了较多的手段来保证数据的可靠性。

① k-safety: VoltDB 的数据是分区的，设定 k-safety（即同步的分区副本数量）参数，可以确保整个集群的稳定性。如果 k 设置为 1，每个分区都在集群内有一个副本，如果有一台机器宕机，整个 VoltDB 集群仍可以正常运行。

② Snapshot（快照）：用户可以根据应用设置快照的周期，VoltDB 可以对数据做快照，然后将其写入磁盘，数据就永久性存在了。如果意外宕机，VoltDB 可以根据磁盘上已有的快照文件，进行数据恢复操作。

③ Command Logging（命令日志）：该功能是 VoltDB 特有的，和快照一起使用，记录保存两次快照之间所有写（write）操作的日志，是为了确保不会有任何事务丢失。读（read）操作不改变数据，所以不被记录。

④ Database Replication：该功能是传统的数据库已有的功能，即在一个物理范围不同的地区建立一个附属集群，并建立一个完全相同的副本，如果主集群因为某些原因不能工作，则立即切换到附属集群上去。

目前已经有较多的企业在使用 VoltDB 数据库。VoltDB 的客户数量已经有 400 多个，合作伙伴也达到 30 个左右，其中包括雅虎、惠普等大型 IT 公司。VoltDB 适合应用在 OLTP 系统中，如金融、零售、Web 2.0 等传统的 OLTP 应用。

9.4 练习题

1. NoSQL 数据库有哪几种存储类型?
2. 什么是列存储?
3. 写一个简易爬虫爬取豆瓣电影资源, 用 MongoDB 存储, 并实现查询接口。
4. 指出 NewSQL 的优点以及可能存在的缺点。

参考文献

- [1] <http://www.thoughtworks.com/insights/blog/nosql-databases-overview>
- [2] Cattell R. Scalable SQL and NoSQL data stores. ACM SIGMOD Record, 2011, 39(4): 12-27.
- [3] Stonebraker M. SQL databases v. NoSQL databases. Communications of the ACM, 2010, 53(4): 10-11.
- [4] Strauch C, Sites U L S, Kriha W. NoSQL databases. Lecture Notes, Stuttgart Media University, 2011.
- [5] <http://zh.wikipedia.org/wiki/Redis>.
- [6] Zawodny J. Redis: Lightweight key/value store that goes the extra mile. Linux Magazine, 2009.
- [7] <http://baike.baidu.com/view/9367705.htm>.
- [8] George L. HBase: the definitive guide. O'Reilly Media, Inc., 2011.
- [9] <http://www.csdn.net/article/2013-04-28/2815090-MemSQL-distributed-version>.
- [10] <http://en.wikipedia.org/wiki/NoSQL>.
- [11] <http://www.sigma.me/2011/06/11/intro-to-nosql.html>.
- [12] <http://zh.wikipedia.org/zh-cn/CAP%E5%AE%9A%E7%90%86>.
- [13] <http://hbase.apache.org/>.
- [14] Corbett J C, Dean J, Epstein M, et al. Spanner: Google's globally distributed database. ACM Transactions on Computer Systems (TOCS), 2013, 31(3): 8.
- [15] Webber, Jim. A programmatic introduction to Neo4j. Proceedings of the 3rd annual conference on Systems, programming, and applications: software for humanity. ACM, 2012.
- [16] <http://www.cnblogs.com/shitouer/archive/2012/06/04/2533518.html>.
- [17] 赵勇, 林辉, 沈寓实, 等. 大数据革命——理论、模式与技术创新. 北京: 电子工业出版社, 2014.
- [18] <http://www.cnblogs.com/justfortaste/p/3532158.html>.
- [19] <http://developers.memsql.com/docs/3.1/concepts/leaf.html>.
- [20] <http://docs.voltDB.com/UsingVoltDB/IntroHowVoltDBWorks.php>.

第10章

大数据与数据挖掘

10.1 数据挖掘的主要功能和常用算法

数据挖掘是在大型数据集中，自动地发现有用信息的过程。数据挖掘技术可以用来探查大型数据库，发现潜在的有用模式，还可以预测未来的样本观测结果。

数据挖掘是一种技术，它将传统的数据分析方法与处理大量数据的复杂算法相结合。数据挖掘为探查和分析新的数据类型以及用新方法分析旧有数据提供了令人振奋的机会。

10.1.1 数据挖掘的主要功能

通常，数据挖掘的任务分为下面两大类。

- ① 预测任务。其目标是根据已知属性的值，预测未知属性的值。
- ② 描述任务。其目标是发现数据中潜在的模式（相关、趋势、聚类、轨迹和异常）。

数据挖掘的主要功能如下。

① 预测建模，以设计变量函数的方式为目标变量建立模型。有两类预测建模任务：分类，用于预测离散的目标变量；回归，用于预测连续的目标变量。两类任务的目标都是训练一个模型，使目标变量预测值与实际值之间的误差达到最小。

② 关联分析，用来发现描述数据中强关联特征的模式。所发现的模式通常用蕴含规则或特征子集的形式表示。由于搜索空间是指数规模的，关联分析的目标是以有效的方式提

取最有趣的模式。

③ 聚类分析,旨在发现密切相关的观测值主群,使得与属于不同簇的观测值相比,属于同一簇的观测值相互之间尽可能类似。

10.1.2 常用算法

1. 关联分析

关联分析主要用于发现隐藏在大型数据集中的有意义的联系,所发现的联系可以用关联规则或频繁项集的形式表示。

进行关联分析时,主要包括两个阶段:第一,从大型数据集中导出频繁项集;第二,利用频繁项集生成关联规则。

(1) 问题定义

在进行关联分析之前,首先要将数据集看成一个个事务的集合,每个事务中包含若干条数据(项)。一个事务表示一个有意义的单元区间,在这个有意义的单元区间中,若干条数据共同出现即称为一个事务。

关联分析的主要任务是从事务集中找出数据之间的强关联关系。这些关系的表现形式有两种:频繁项集和关联规则。其中关联规则是我们最终需要的输出,但是关联规则很难直接从数据集中得到,为了解决这个问题我们首先需要从数据集中找出关联的另外一种表现形式——频繁项集,然后将频繁项集转化为关联规则。这是基于这样一种思想:经常在一起出现的事物间往往存在一定的关系。因此问题就转化为寻找经常在一起出现的数据子集。相关概念定义如下。

项:在关联分析中,我们将数据集看成事务的集合,每个事务又看成不同数据的集合,数据集中的每个数据称为一项,可见项之间是不相同的。

项集:在一起出现的项的集合。一个项集可以表示一个事务,项集的宽度定义为项集中项的个数。

频繁项集:出现次数在一定阈值之上的项集,称为频繁项集。可以通俗地理解为,某几类数据经常一起出现,这些数据就构成一个频繁项集。

项集的支持度计数:项集的支持度计数是包含特定项集的事务个数。如果项集在某一事务中出现,则项集的支持度计数加一。

关联规则:关联规则是形如 $x \rightarrow y$ 的表达式,其中 x 和 y 是不相交的项集。

支持度:又称关联规则的支持度,指数数据集(事务集)中包含该项集的事务数占总事务数的百分比。

可信度:又称关联规则的可信度,是 x 和 y 的并集项集支持度与 x 项集支持度的比值。

对于一个 $x \rightarrow y$ 规则,可信度和支持度定义如下。

规则的支持度:

$$s(x \rightarrow y) = \frac{\sigma(x \cup y)}{N}$$

规则的可信度:

$$c(x \rightarrow y) = \frac{\sigma(x \cup y)}{\sigma(x)}$$

借由可信度和支持度的概念,我们可以给出关联规则挖掘问题的形式描述如下:给定事务集合 T ,关联规则发现是指找出支持度大于等于最小支持度阈值,且可信度大于等于最小可信度阈值的所有规则。

最朴素的关联规则挖掘算法就是找出所有的规则,分别计算出它们的支持度和可信度,但是这种算法的计算量惊人,完全不可行。通过可信度的定义公式我们可以发现,可信度可由支持度计算得到,而支持度又与项集的大小相关联。因此问题转化为寻找满足一定大小的项集,通常项集越大,支持度越高。因此我们需要寻找一个尽可能大的项集,即寻找频繁项集。

寻找频繁项集的朴素方法是遍历所有可能出现的项集组合,然后分别记录它们在事务集中出现的次数,再选出出现次数较多的项集。这种暴力搜索方法的复杂度是指数级的,同样不可行,必须对搜索算法进行改进。下面介绍两种常用的频繁项集发现算法。

(2) Apriori 算法

Apriori 算法的主要思想基于如下原理:如果某一项集是频繁的,那么它的子集也是频繁的;如果一个项集是非频繁的,那么它的所有超集也是非频繁的。使用该原理可以避免项集数目的指数增长,具体来说,如果我们发现一个较小的项集是非频繁的(不满足支持度阈值的要求),那么它的所有超集我们都不需要考虑了,从而大大缩小了搜索空间。另外,原理的前半部分告诉我们,频繁项集实际上是由更小的频繁项集组合而成的(并集),因此如果我们能够找出某个特定大小的全部频繁项集,我们就可以使用它们构建出全部的更大的频繁项集。最后,必须指出,越大的频繁项集,数目越少,因此最后一定存在一个最大的频繁项集。

Apriori 算法的大体步骤如下:首先,搜索整个事务集,找出所有大小为 $K=1$ 的频繁项集,由于项集的大小只有 1,所以这个过程是可行的;然后是一个迭代的过程,每次迭代都会得到一个大小为 K 的频繁项集列表,这个列表中包含了所有大小为 K 的频繁项集,在下次迭代中使用大小为 K 的频繁项集构建出大小为 $K+1$ 的频繁项集;最后,当无法找到更大的频繁项集时,算法停止。

我们可以将 Apriori 算法分成两个部分,第一部分负责由大小为 K 的频繁项集产生大小为 $K+1$ 的候选项集(注意,所有大小为 $K+1$ 的频繁项集一定在候选项集中,但候选项集并非都是频繁项集),第二部分负责检查大小为 $K+1$ 的候选项集是不是频繁项集。

生成 $K+1$ 候选项集部分的算法如下:首先对比 K 频繁项集(每个项集中的元素都进行了排序)中所有项集的前 $K-1$ 个元素,如果两个项集的前 $K-1$ 个元素完全相同,则合并这

两个集合，从而得到一个大小为 $K+1$ 的候选项集。

检测候选项集是否为频繁项集的算法比较简单，只须扫描事务记录即可。

最后从 $K=1$ 开始反复先后迭代这两部分算法，即可得到所有的频繁项集。

(3) FP-Growth 算法

FP-Growth 算法是 Apriori 算法的一种改进形式。在 Apriori 算法中，在检测候选项集阶段，对于每个频繁项集，算法都会扫描整个数据集以判断候选项集是否频繁，从而拖慢了算法的速度。其实这其中有很多扫描是不必要的，候选项集之间往往有很多项是重合的，这些重合部分的判断是可以重用的。FP-Growth 算法正是将判断过程组织成树的形式，从而重用了重合部分的判断，它只需要对数据库进行两次扫描，大大提高了算法的运行速度。

FP-Growth 算法将数据频度信息存储在被称为 FP 树的数据结构中，如图 10.1 所示。

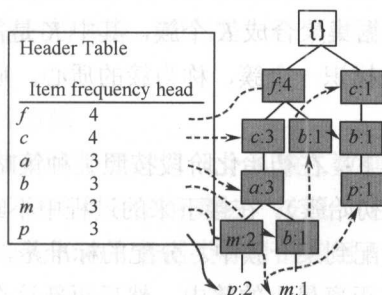


图 10.1 FP 树示例

一个元素可以在 FP 树中出现多次，各同名节点间使用节点链接连接起来。树中的每条路径对应一个项集，树中的每个节点存储了在相应项集中的频数。从纵向上来看，路径与频繁项集对应，从横向上来看各同名节点间构成一条同名节点链。FP 树还包括一个头指针表，头指针表中保存了每个频繁项的同名节点链的头指针和该频繁项的节点链计数和。

FP-Growth 算法的工作流程如下：

- ① 扫描整个数据集，对所有项计数。
- ② 第二次扫描数据集，构建 FP 树，这次扫描只针对频繁的项进行。

在这里我们主要关注第二步构建 FP 树的过程，首先利用第一步得到的频繁项对各事务进行过滤，将非频繁项从事务中删去，然后将事务中的项按照频繁度由高到低进行排序。最后将每个事务一次添加到树中，每个事务对应一条树中的路径，若路径上的节点已存在，则增加该节点的计数值；若不存在，则添加一个分支。同时要注意生成头指针表和同名节点链接。

生成了 FP 树之后，我们就可以从 FP 树中挖掘出频繁项集。首先针对某一频繁项回溯其前缀路径，然后用前缀路径构成一棵条件 FP 树，从条件 FP 树中裁剪掉非频繁项，将剩余部分当作新的 FP 树，递归进行挖掘即可。

(4) 规则生成

在得到了所有满足阈值的频繁项集之后，需要考虑如何从频繁项集中得到关联规则。

构建的关联规则要满足一定的可信度。采用的方法与频繁项集发现类似，首先产生箭头右侧为一个元素的关联规则，然后滤除那些不满足最小可信度的关联规则，接着生成箭头右侧为两个元素的关联规则，最后当箭头右侧无法再增加元素时，算法停止。

每次生成规则的基本方法是，从频繁项集中拿出若干元素到箭头右侧，然后判断可信度是否满足要求。

2. 聚类分析

聚类分析将数据划分成有意义或有用的组（簇）。如果目标是划分成有意义的簇，则簇应当捕获数据的自然结构。然而，在某种意义上，聚类分析只是解决其他问题的起点。无论是理解数据还是数据预处理领域，聚类分析都扮演着重要角色。

（1） K 均值

K 均值算法可以将给定数据集聚合成 K 个簇，其中 K 是需要用户指定的参数。 K 均值算法使用簇中所有点的中心来标识一个簇，称为簇的质心。质心的坐标，采用所有点坐标的平均值来确定。

K 均值算法的大致过程如下：在初始化阶段按照某种策略（通常是随机）选取 K 个数据点作为初始质心（同时也是初始簇）。在接下来的过程中不断地进行迭代，每次迭代将数据集中某个未被分配簇的点分配到某一簇中。分配的标准是，计算这个点到 K 个质心的距离，然后将这个点分配到与它距离最小的簇中。然后更新这个有新成员加入的簇，即重新计算该簇的质心。以此不断迭代，直到所有的点都被分配了一个簇为止。

前面提到， K 值是由用户指定的，不同的 K 值会对最后的聚类效果产生影响。为了衡量聚类结果的好坏，在这里使用误差平方和（SSE）作为聚类结果的指标。SSE 是指一个簇中各点到质心距离的平方和，SSE 越小则聚类效果越好。

一种对聚类结果优化的方法是，在聚类结束之后，将 SSE 最大的簇划分为两个簇（即在该簇上运行 $K=2$ 的 K 均值算法），为了保持 K 不变，再将某两个簇合并，可以合并距离最小的质心，也可以合并两个使 SSE 增加最少的质心。

基于上述优化方法给我们带来的启示，我们可以对 K 均值算法进行改良，得到二分 K 均值算法。

二分 K 均值算法的思想是，首先将所有数据看成一个簇，然后将该簇分裂为两个，再选择其中一个簇进行分裂，像这样每次选择一个簇进行分裂，每次都会使簇的总数加一，直到簇的数量达到 K 为止。每次选择分裂簇的标准是，选择那个分裂之后可以最大程度降低 SSE 的簇，也可以采用分裂 SSE 最大的簇的策略。分裂簇的方法是，在该簇上运行 $K=2$ 的 K 均值算法。

（2）层次聚类

层次聚类算法与 K 均值聚类算法的方向相反。首先，它将每个数据点看成一个单独的簇，然后按照一定的策略将这些簇进行合并，直到达到停止条件为止。

可能的停止条件有，簇的数目达到一个定值；或者聚成一个大簇；或者再继续进行合

并，会产生一个明显错误的簇。

在合并过程的每一次迭代中，我们选择距离最近的两个簇进行合并。簇之间的距离依然用簇质心间的距离度量。也可以采用一些其他的合并策略：定义簇的距离为簇中所有点之间的最短距离；定义簇距离为两个簇间所有点对距离的平均值；合并半径最小的两个簇，簇的半径定义为簇内点到质心的最大距离；合并直径最小的簇，簇的直径定义为簇中两点之间的最大距离。

3. 分类

分类任务的输入数据是记录的集合。记录也称实例或样例，用元组 (x, y) 表示，其中 x 是属性的集合，而 y 是一个特殊的属性，指出样例的类标号（也称分类属性或目标属性）。属性主要是离散的，但属性集也可以包含连续的特征。另一方面，类标号却必须是离散属性，这正是区别分类与回归的关键。回归是一种预测建模任务，其中目标属性 y 是连续的。

(1) 解决分类问题的一般方法

从整体来看，分类问题实际上是要建立一个从输入数据到分类标签的映射。机器学习建立这个映射模型的方法是，使用某种学习算法，按照一定的策略对输入数据进行分析，找出一个能够很好拟合输入数据和输入数据类标号的映射，同时这个映射还能够正确地预测未知数据和它的类标号。

对于任何有监督的机器学习而言，首先都必须有一个我们已经很了解的训练数据集。具体来说，在分类问题中，数据相当于问题求解的条件，每条数据对应的标签是我们想要得到的答案。现在有一个训练数据集，这个数据集中有许多数据，同时知道这些数据对应的分类标签。现在我们需要从这些已知的数据和数据对应的标签中，找出数据和标签的规律，一旦找出这个规律，我们就可以对那些我们不知道标签的数据进行预测，预测它们的标签是什么。寻找规律的过程不是盲目的，必须遵循一定的准则和方法。也可能同时有多个规律可以解释已知数据和标签的对应关系。好的规律不仅能够正确概括已知数据和标签的对应关系，而且面对未知的数据时也能够给出准确的预测。

在上面的叙述中，已知数据和标签称为训练集，规律称为模型，寻找规律的准则和方法称为分类算法，未知数据称为检验集，正确概括已知数据的能力称为准确性，预测未知数据的能力称为泛化性。

下面将具体介绍各种分类算法。

(2) 决策树

决策树是一种易于解释且应用广泛的分类算法。决策树的基本思想是构建一系列检验数据属性的 if-then 规则，通过不断地在数据上应用这些规则来逐步对数据进行细分。每个规则相当于一个问题，用规则来检验数据集上的数据，根据每条数据对规则的不同响应可以将数据集分为几个子集。再在子集上递归应用合适的规则，不断地对数据集进行细分，当某个子集中只剩下同一类数据时，即得到分类的最终答案——数据的分类标签。这种递

归的不断分支的结构可以用树来表示，所以称为决策树。一个比较通俗的决策树示例如图 10.2 所示。

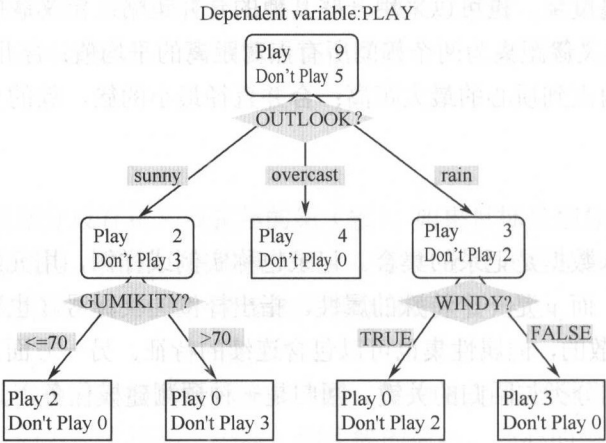


图 10.2 决策树示例

决策树有两大优点：

- ① 决策树模型可读性好，具有描述性，有助于人工分析；
 - ② 效率高，一次构建，反复使用，每一次预测的最大计算次数不超过决策树的深度。
- 决策树的一大缺点是容易对数据形成过拟合。

如果将决策树生成过程视为根据数据的某一属性对数据集进行不断划分的过程，那么决策树生成过程必须解决如下几个问题：在哪个属性上划分，划分的标准是什么，何时结束划分。这三个问题的不同解决方案即构成了不同的决策树算法。

对于前两个问题目前主流的算法都选择熵和信息增益及其比值作为属性选择和划分的标准。

熵是一种反映数据类别一致程度的数学概念，熵的定义如下：

$$H(X) = -\sum_{i=1}^n p_i \log p_i$$

$$H(Y|X) = -\sum_{i=1}^n p_i H(Y|X = x_i)$$

当数据集中数据类别相同时，熵为 0。对于二分类问题，当数据集中两种类别的数据数量相同时，熵达到最大。

信息增益是指当数据集按某种规则划分后熵变化的程度，计算公式如下：

$$g(D, A) = H(D) - H(D|A)$$

信息增益比定义为信息增益与关于该特征的熵的比值，公式如下：

$$g_R(D, A) = \frac{g(D, A)}{H_A(D)}$$

其中， $H_A(D) = -\sum_{i=1}^n \frac{|D_i|}{|D|} \log_2 \frac{|D_i|}{|D|}$

有了这些指标的基础,接下来介绍两种决策树生成算法。

ID3: ID3 算法的核心是选用信息增益作为决策树每层划分的依据。具体来讲,从根节点开始,分别计算按各个特征划分的信息增益,然后在信息增益最大的特征上进行划分。由该特征的不同取值对数据集进行划分。在子节点上递归地进行划分,直到再进行进一步划分时,信息增益小于某一阈值为止。

C4.5: C4.5 算法与 ID3 算法相似,C4.5 算法在生成过程中使用信息增益比作为划分的依据。

决策树的一个缺点是,容易对数据进行过拟合。为了避免过拟合,需要对决策树进行简化,对决策树的简化称为剪枝。剪枝有两种策略:先剪枝和后剪枝。先剪枝是指在决策树生成过程中,当划分指标小于某一阈值时,停止决策树的生长。后剪枝是指先让决策树完全生长,然后自底而上地修剪决策树。后剪枝有两种方式:第一,用子树下最多类作为类标签节点来代替子树;第二,用子树中主要分支来代替子树。

(3) 最近邻分类器

最近邻分类器是分类算法中思想最简单的一种算法,也是最接近直觉的算法。它的优点在于它是一种消极学习算法,不需要对数据集建立模型,从而也就不需要对数据集进行过多的假设。但同时消极学习也带来了许多缺点,比如对噪声敏感、对数据预处理要求高以及计算开销过大等。

最近邻算法的基本思想是,目标数据的类别往往和与它相似的数据相同。基于此,我们寻找 K 个与目标数据最相似的已知数据,根据这些已知数据的类别标签对目标数据的标签进行推断。

我们将数据视为 N 维空间中的点,数据之间的相似度用点之间的距离来表示。点之间的距离有多种定义方法,最常用的就是几何上的欧式距离。

在找到离目标数据最近的 K 个样本点之后,可以根据这 K 个样本点的类标签对类别进行多数表决:

$$y' = \arg \max_v \sum_{(x_i, y_i) \in D_z} I(v = y_i)$$

其中, v 是类标号; y_i 是一个最近邻的类标号; $I(\cdot)$ 是指示函数,如果其参数为真,则返回 1,否则返回 0。

这 K 个点到目标点的距离有近有远,距离越近的点,在判别类别时越有说服力。我们可以根据距离对类别做加权平均:

$$y' = \arg \max_v \sum_{(x_i, y_i) \in D_z} w_i \times I(v = y_i)$$

其中, w_i 是距离相关的权值, $w_i = 1/d(x', x_i)^2$ 。

(4) 朴素贝叶斯分类器

朴素贝叶斯分类器是一种基于贝叶斯定理和特征独立假设的分类方法。它是一种生成模型,即它学习的不是如何判别数据的类别,而是学习产生数据的概率分布,再基于此分布,由贝叶斯定理求出数据从属于各类别的后验概率,从各后验概率中选取一个最大的类

别作为输出。朴素贝叶斯分类器的实现方法相对简单，学习和预测效率较高。

在了解朴素贝叶斯分类器之前，简单介绍一下贝叶斯定理。从实用角度出发，我们可以将数据产生的过程描述为，某一个从属于类别 Y 的数据，它在观测中表现出的值是 X 。我们可以将这个过程表示为如下公式：

$$P(X, Y) = P(Y | X) \times P(X) = P(X | Y) \times P(Y)$$

从整个数据集来看，其中某个数据从属于 Y 类有一个概率，而该类别又以一定的概率表现出观测值 X ，因此数据属于 Y 类且表现出观测值 X 的概率为二者的乘积。

但是，对数据进行分类的过程是上述过程的逆过程。我们是根据观测值 X 来预测数据属于哪一类，这一概率可以用贝叶斯公式计算：

$$P(Y | X) = \frac{P(X | Y)P(Y)}{P(X)}$$

为了计算贝叶斯公式中的各项，我们往往需要对各项的概率分布做出某种假设。在朴素贝叶斯分类器中，我们假设属性之间是相互独立的，即有如下公式：

$$P(X | Y = y) = \prod_{i=1}^d P(X_i | Y = y)$$

代入贝叶斯公式，有

$$P(Y | X) = \frac{P(Y) \prod_{i=1}^{a'} P(X_i | Y)}{P(X)}$$

因此，我们只要计算出各类别在相应属性上表现出相应概率的值即可。在实际中，我们利用某一类别中在某一属性上取相应值的样本比例来估计这一概率。至此，朴素贝叶斯分类器的构建方法如下：

- ① 求出每个类别中各个属性取各个值占整个类别数量的比值，这个比值将用来预估概率。
- ② 对于目标数据，每个属性必然取一个值，在上面得到的概率集中查找该属性取该值在各个类别中的概率。
- ③ 将步骤②中得到的概率按类别分组，每组内概率相乘。
- ④ 取步骤③中最大的概率类别作为结果输出。

(5) LR

LR (Logistic Regression) 虽然是一种回归方法，但它往往被用来进行分类。LR 模型的训练速度相对较快，是广泛应用的一种分类方法。

LR 模型的核心是 sigmoid 函数 (图 10.3)，该函数取值在 0 到 1 之间。LR 的基本策略是从数据集中为每个类别学习一个参数，利用该参数生成一个 sigmoid 函数，用 sigmoid 函数在相应数据点上的取值预估该数据被分为该类的概率。

sigmoid 函数从本质上来讲是一种概率乘积模型，概率的定义如下：

$$\text{odd} = \frac{p}{1-p}$$

其中， p 是事件的概率。

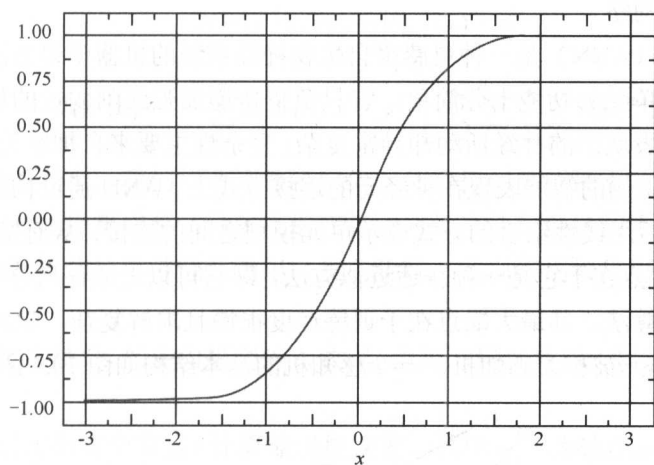


图 10.3 sigmoid 函数

将概率转化为对数的好处是，可以将取值区间扩展到整个正实数集，从而降低取值上的约束。如果我们将数据的各个属性视为相互独立的，各个属性的概率之间的复合可以用概率积来表示。为了简化乘积运算，我们可以对概率取对数，这样就可以将乘法转化为加法，这时的概率称为对数概率：

$$\log \text{it}(p) = \log \frac{p}{1-p}$$

LR 的核心思想就是，利用输入的一个线性组合去拟合对数概率的和，即每一项有

$$\log \frac{P(Y=1|x)}{1-P(Y=1|x)} = w \cdot x$$

反解概率，则得到 sigmoid 函数：

$$P(Y=1|x) = \frac{1}{1+e^{-w \cdot x}}$$

其中只有线性加权系数 w 未知，LR 正是要从数据集中学习该参数，对于二分类来讲，是最小化分类误差：

$$L(w) = \sum_{i=1}^N y_i \log P(Y=1|x) + (1-y_i) \log P(Y=0|x) = \sum_{i=1}^N y_i (w \cdot x_i) - \log(1 + e^{-w \cdot x_i})$$

可以使用梯度下降法求解。

综上所述，使用 LR 的步骤如下：

- ① 对每个类别构建一个目标函数；
- ② 用梯度下降法（或其他方法）求解；
- ③ 选取各类别中有最大的 sigmoid 函数的类别作为结果输出。

（6）人工神经网络

人工神经网络（ANN）是一种灵感来自生物神经系统的机器学习方法。在生物神经系统中，基本的神经单元的功能十分简单，它只负责传递加强或削弱过的信号。但是由多个简单神经单元联合表现出的神经活动却异常复杂，复杂性主要来自神经元之间的连接方式，因此可以这样说，生物的智能表现在神经元的连接方式上。ANN 通过构建一种相对简单的数学单元模型，然后用线性组合的方式表示单元模型之间的连接，从而模拟自然神经网络。

ANN 的最大优点在于它是一种普适近似方法，即它可以近似任何分类函数，而且是人工假设最少的一种算法。其最大缺点在于训练速度很慢且求解复杂，学习过程难以解释。

ANN 的基本单元被称为感知机，一个感知机的基本结构如图 10.4 所示。

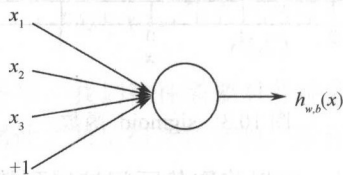


图 10.4 感知机的基本结构

感知机主要由两部分构成：第一部分对输入进行线性加权；第二部分称为响应函数，它利用第一部分的输出作为输入，通过一个函数产生输出。ANN 之所以能够模拟多种分类函数，一大原因就是响应函数的取值比较灵活。

有了单元模型，下一步就是以合理地方式将这些简单单元连接起来。在 ANN 中，通常采用乘积连接结构，每一层之间的神经元都有连接，用加权的权重来代表连接的强弱。由于增加层数会大幅提升求解难度，因此通常采用输入层、隐藏层、输出层的三层神经网络。多层感知机如图 10.5 所示。

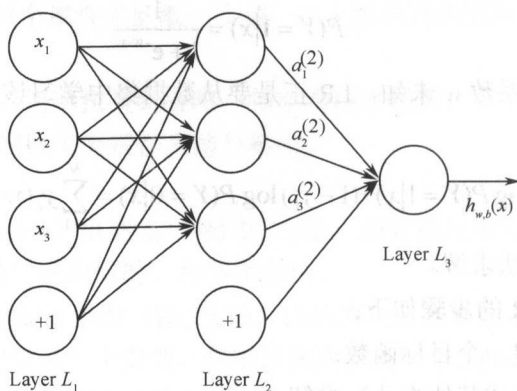


图 10.5 多层感知机

对于一个分类问题，分类结果由输出层表示，通常以分类误差的平方作为目标函数：

$$J(W, b: x, y) = \frac{1}{2} |h_{w,b}(x) - y|^2$$

$$J(W, b) = \left[\frac{1}{m} \sum_{i=1}^m J(W, b: x^{(i)}, y^{(i)}) \right] + \frac{\lambda}{2} \sum_{l=1}^{n_l-1} \sum_{i=1}^{s_l} \sum_{j=1}^{s_{l+1}} (W_{ji}^{(l)})^2$$

为了使分类误差最小, 我们需要调整各权值, 使得目标函数最小, 这通常通过反向传播算法来实现。反向传播算法是一种多次迭代算法, 每次迭代包括两个阶段: 前向阶段, 使用上次迭代得到的权值计算神经元输出; 反向阶段, 反向应用权值更新公式产生新的权值。具体如下:

① 随机初始化各参数, 然后向模型输入数据, 由第二层到最后一层逐层计算激励函数, 直到输出层。

② 对于输出层 n_l 的每个节点 i 计算输出层误差, 其中 $z_i^{(n_l)}$ 为输出层 n_l 的第 i 个节点的激励函数输入:

$$\delta_i^{(n_l)} = \frac{\partial}{\partial z_i^{(n_l)}} \frac{1}{2} |y - h_{w,b}(x)|^2 = -(y_i - a_i^{(n_l)}) \cdot f'(z_i^{(n_l)})$$

③ 由输出层向前逐层计算误差, 对于每一层 $l = n_l - 1, n_l - 2, \dots, 2$ 的第 i 个节点, 由其后一层的误差值可以计算出当前层的误差值:

$$\delta_i^{(l)} = \left(\sum_{j=1}^{s_{l+1}} W_{ji}^{(l)} \delta_j^{(l+1)} \right) f'(z_i^{(l)})$$

④ 由误差值计算偏微分:

$$\frac{\partial}{\partial W_{ij}^{(l)}} J(W, b: x, y) = a_j^{(l)} \delta_i^{(l+1)}$$

$$\frac{\partial}{\partial b_i^{(l)}} J(W, b: x, y) = \delta_i^{(l+1)}$$

⑤ 更新参数值:

$$W_{ij}^{(l)} = W_{ij}^{(l)} - \alpha \frac{\partial}{\partial W_{ij}^{(l)}} J(W, b)$$

$$b_i^{(l)} = b_i^{(l)} - \alpha \frac{\partial}{\partial b_i^{(l)}} J(W, b)$$

(7) 支持向量机

支持向量机 (SVM) 是目前最为成熟、应用最为广泛且具有坚实统计学理论基础的机器学习方法。它拥有理论上的最佳分类效果, 可以很好地应用于高维数据, 分类速度很快, 同时通过改变其核函数, 可以灵活地解决一些线性不可分问题。

对于一个二分类问题, 分类的本质是找出一个超平面将两类点分割在超平面两侧, 这样的超平面可以有很多个。SVM 的优势在于, 它可以在诸多超平面中找出最佳的一个。

图 10.6 显示了超平面和最佳超平面的区别。

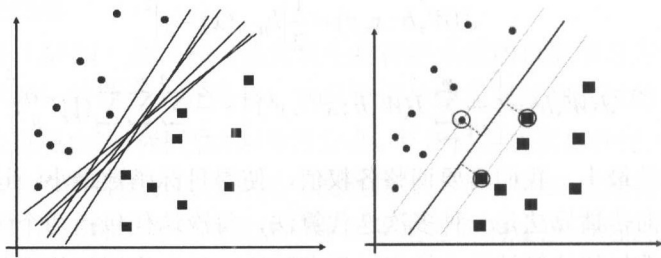


图 10.6 超平面和最佳超平面对比图

最佳超平面一般定义为，两类点中距离超平面最近的点的距离和最大的超平面。与超平面平行且经过距离超平面最近的两类点的平面称为决策边缘。对于二分类问题，决策边缘有两个，最佳超平面的最佳之处在于它可以使决策边缘之间的间隔最大化。越大的间隔意味着越强的泛化能力，即使数据受到了扰动，落在了间隔内部，SVM 依然能够正确分类。

既然 SVM 寻找的是间隔最大化的超平面，那么目标函数自然就是间隔，而间隔表示为离超平面最近的点，同时还要满足所有已知样本都要正确分类的约束条件，利用拉格朗日乘数法得到目标函数如下：

$$L_P = \frac{1}{2} \|w\|^2 - \sum_{i=1}^N \lambda_i (y_i (w \cdot x_i + b) - 1)$$

利用 KTT 条件将不等式约束转化为等式约束，得到对偶目标函数：

$$L_D = \sum_{i=1}^N \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j x_i \cdot x_j, \quad 0 \leq \lambda_i$$

这是一个二次规划问题，可以使用 SMO 算法求解。

在一些情况下，我们希望牺牲一些分类的准确率，而换取更强的泛化能力。如图 10.7 所示，虽然右侧的平面错分了一些点，但是泛化能力大大提高，而错分的点很可能是数据有噪声造成的。

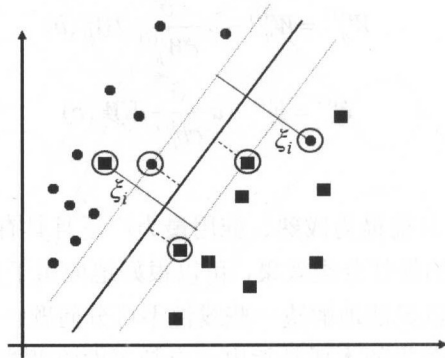


图 10.7 容忍噪声的超平面

为了达到这一效果，我们需要在目标函数中修改边界间隔，同时为了避免间隔无限扩

大, 还需要引入间隔惩罚项, 具体公式如下:

$$L_P = \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^N \xi_i - \sum_{i=1}^N \lambda_i (y_i (\mathbf{w} \cdot \mathbf{x}_i + b) - 1 + \xi_i) - \sum_{i=1}^N \mu_i \xi_i$$

KTT 条件处理后:

$$\begin{aligned} L_D &= \sum_{i=1}^N \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j \\ \sum_{i=1}^N \lambda_i y_i &= 0 \\ 0 &\leq \lambda_i \leq C \end{aligned}$$

用 SMO 算法求解即可。

对于不能线性分类的问题, SVM 的处理方法是, 利用一个函数将不可分的数据映射到高维空间中, 若数据在高维空间中线性可分, 即可在高维空间中使用 SVM。

非线性的 SVM 学习任务与线性 SVM 相似, 只是这时需要代入的是核函数映射之后的点:

$$L_D = \sum_{i=1}^N \lambda_i - \frac{1}{2} \sum_{i,j} \lambda_i \lambda_j y_i y_j \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x}_j)$$

合法的核函数必须满足 Mercer 定理。

常用的核函数如下。

线性核函数: $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j$

多项式核函数: $K(\mathbf{x}_i, \mathbf{x}_j) = (1 + \mathbf{x}_i^T \mathbf{x}_j)^p$

高斯 (径向基) 核函数: $K(\mathbf{x}_i, \mathbf{x}_j) = e^{-\|\mathbf{x}_i - \mathbf{x}_j\|^2 / 2\sigma^2}$

最后总结 SVM 的一般方法:

- ① 合理使用核函数对数据进行预处理, 若数据已经线性可分, 则无须使用核函数;
- ② 选取适当的容错因子;
- ③ 求解目标函数。

4. 回归

分类可以简单地看成由已知数据学习模型来预测离散标签的方法, 而回归则是预测连续数值的方法。回归的主要目的是预测数值型数据, 最直接的方法就是根据输入数值写一个计算目标值的公式。这个目标公式被称为回归方程, 回归的主要方法就是优化回归方程中的参数, 从而减小回归预测的误差。

(1) 简单线性回归

如果回归方程是输入数据属性值的线性加权, 那么这种回归就被称为线性回归。线性回归需要确定的系数是各个属性对应的权值。回归方程可以写为

$$y = \mathbf{x}^T \mathbf{w}$$

线性回归的常用学习方法是最小二乘法, 普通最小二乘法 (OLS) 的思想是, 从训练集中学习一组回归系数, 以回归误差的平方和作为目标函数, 优化目标函数, 使目标函数

最小化,从而使误差最小化。目标函数如下:

$$\sum_{i=1}^m (y_i - x_i^T w)^2$$

对回归系数求导,令导数为 0,得出回归系数:

$$\hat{w} = (X^T X)^{-1} X^T y$$

(2) 局部加权线性回归

简单线性回归只用一条直线拟合数据,模型表现力有限,在实际应用中很少使用。一种改进的方法是,用多条线段构成一条折线来对数据进行拟合,即在不同的数据段上分别构建简单线性回归模型。这种改进被称为局部加权线性回归(LWLR)。

在 LWLR 中,我们给待预测点附近的每一个点赋予一个权值。在这个待预测点和它的邻居构成的子集上,我们构建简单线性回归模型。每次在进行预测之前,都需要事先选取相应的数据子集。目标函数与简单线性回归相同,回归系数如下:

$$\hat{w} = (X^T W X)^{-1} X^T W y$$

其中, w 是每个邻居数据点的权值。权值可以使用“核”来计算,常用的权值核是高斯核,它对距离目标数据点近的数据点赋予较大的权值,对距离目标数据点远的数据点赋予较小的权值,高斯核如下:

$$w(i, j) = e^{-\frac{|x^{(i)} - x^{(j)}|^2}{2k^2}}$$

局部加权线性回归具有较高的精确度,但是为了做出预测,必须保存所有的训练数据。

(3) 岭回归

如果数据的特征比数据样本点还多,前面所述的线性回归算法在求解目标函数时,将遇到矩阵不可逆的情况,从而使求解无法进行。为了解决这个问题,我们引入岭回归算法。

岭回归的思想十分简单,为了使 $X^T X$ 矩阵变得可逆,我们在其上加一个 I 矩阵,得到的新矩阵一定是可逆的。在这种情况下,回归系数求解公式变为

$$\hat{w} = (X^T X + \lambda I)^{-1} X^T y$$

岭回归诞生之初是为了解决特征过多而样本过少的问题,现在也用于在回归过程中引入误差,从而防止过拟合。由于新加入的矩阵限制了回归系数的和的大小,新加入的矩阵又被称为回归系数的惩罚项,它能够减少不必要的参数,因此也被称为缩减系数。

为了加快求解系数,我们可以证明,在 OLS 中添加如下约束后,简单线性回归会变为岭回归:

$$\sum_{k=1}^n w_k^2 \leq \lambda$$

进一步简化约束条件,我们可以得到 Lasso 方法。Lasso 方法的约束条件如下:

$$\sum_{k=1}^n |w_k| \leq \lambda$$

在 Lasso 约束下,回归系数会更加接近 0,这意味着,在集中约束条件中,Lasso 的效

果最好。但是必须指出,求解 Lasso 约束极大地增加了计算难度。为了减轻计算压力,可以使用前向逐步回归方法。

前向逐步回归方法得出的解的质量与 Lasso 差不多,但计算更为简单。它的求解过程类似梯度下降法,并且它是一种贪心算法,即每一步都尽可能减小预测误差。在初始化阶段,所有权值被初始化为 1,然后每次迭代中,对每个权值增大或减少一个很小的值,如果改变权值之后,误差变小,则使用改变之后的权值作为新的权值。

(4) 分类回归树

线性回归是一种强大的分析模型,但是线性回归的拟合能力有限,很难建立全局的线性回归模型。为了充分发挥线性回归的作用,必须将整体数据集切分为局部数据集,再在局部数据集上建立回归模型。

决策树是切分数据集的有效方法,但前面介绍的决策树模型只能对标称型数据进行切分,如果要使用决策树对数值型数据进行划分,就不得不对决策树进行改进。

分类回归树(CART)是一种使用改进的决策树对数据集进行划分,并在划分数据集上分别运用线性回归的算法。它可以被视为回归方法和决策树的结合。

CART 的每个数据子集的回归方法可以使用前面所述的回归方法,在此不再赘述。我们主要介绍如何改进决策树算法。

之前使用的 ID3 和 C4.5 决策树生成算法存在的第一个问题是,切分过于迅速。在每次切分之后,用来切分的特征值将不再起作用,即每个子节点中,该特征的值相同。如果想要在子节点上运行回归算法,我们不希望看到这种情况,因为这会使某个回归系数完全失效。第二个问题是, ID3 和 C4.5 算法使用熵的衍生指标作为切分标准,对于数值型数据,计算熵十分困难。为了解决第一个问题, CART 使用二元切分法来进行切分,从而防止切分速度过快,同时二元切分也易于在数值型数据上使用。对于第二个问题, CART 使用总方差来替代熵指标,作为切分判断的标准。

该算法的过程大致如下:

- ① 对每个特征的每个特征值,将数据集切分成两部分;
- ② 计算切分误差;
- ③ 若当前误差小于最小误差,则将这个切分设为最佳切分,用当前误差更新最小误差;
- ④ 在每个特征的每个特征值上迭代上述过程。

与决策树一样, CART 算法同样需要进行剪枝,从而避免过拟合。剪枝同样可分为先剪枝和后剪枝两种。但在 CART 中,后剪枝的效果明显好于先剪枝。在这里着重介绍后剪枝。

后剪枝的方法如下:

- ① 对于已有的树,如果存在一个子树,则在子树上递归后剪枝过程;
- ② 计算合并当前两叶子节点误差和不合并的误差;
- ③ 若合并后误差降低,则合并。

如果需要在这个改进的决策树中加入回归算法,只需要改进 CART 中的计算方法即可,即在每个需要计算节点误差的步骤中,在该节点上运行线性回归算法,用回归预测误差取代节点误差。

10.2 大数据时代的数据挖掘

随着数据库技术的迅速发展及数据库管理系统的广泛应用,人们积累的数据越来越多,正在从 TB 级往 PB 级发展。目前的数据库系统已经从单机模式发展为分布式模式,从结构化模式发展为非结构化模式,从关系型数据库(RDBMS)发展为非关系型数据库(NoSQL),并可以高效地实现数据的录入、查询、统计等功能。但由于计算能力有限,以及数据挖掘算法迭代复杂,难以发现数据中存在的关系和规则,或根据现有的数据预测未来的发展趋势,缺乏挖掘数据背后隐藏的知识的手段,导致了“数据爆炸但知识贫乏”的现象。

10.2.1 传统数据挖掘解决方案

计算能力和存储能力有限,以及数据挖掘算法迭代复杂等原因,决定了传统的数据挖掘方式不可能建立在元数据之上,因为任何数据的元数据本身往往都在 TB 或 PB 级,传统的数据挖掘一般都使用采样的方式来获取样本,因此样本的覆盖率和分布情况显得十分重要,而覆盖率高且分布情况符合元数据的采样几乎是不可能获得的;其次为了提高准确率,传统数据挖掘往往都在数据挖掘算法和模型上做文章,因而都采用较复杂的数据挖掘算法和较复杂的模型来提高性能。

这其中对算法支持比较好且比较典型的工具有:由美国 MathWorks 公司开发的商业数学软件 MATLAB,由新西兰怀卡托大学用 Java 语言开发的数据挖掘常用软件 Weka,由新西兰奥克兰大学的 Ross Ihaka 和 Robert Gentleman 联合开发的 R 语言,以及由 IBM 公司开发的一系列用于统计学分析运算、数据挖掘、预测分析和决策支持的统计产品与服务解决方案 SPSS 等。

10.2.2 分布式数据挖掘解决方案

随着分布式计算技术、云计算技术、Hadoop 生态圈、内存数据库、非结构化数据库等技术的发展,以及对大数据挖掘的需求,一批分布式数据挖掘解决方案涌现出来,其中比较典型的有由 Apache 推出的基于 Hadoop 的 Mahout,以及由加利福尼亚大学伯克利分校 AMP 实验室推出的基于 Spark 的 MLBase。

1. Mahout

Mahout 是 Hadoop 家族中与众不同的一个成员，是一个基于 Hadoop 的机器学习和数据挖掘的分布式计算框架。Mahout 是一个跨学科产品，同时也是 Hadoop 生态圈中非常有潜力的一个项目。Mahout 为数据分析人员解决了大数据的门槛问题，为算法工程师提供了基础的算法库，为 Hadoop 开发人员提供了数据建模的标准。

Mahout 是基于 Hadoop 的分布式数据挖掘平台，提供了具备可扩充能力的机器学习类库。通过和 Apache Hadoop 分布式框架结合，Mahout 能使用分布式系统来实现高性能计算。如果说 Hadoop 是大数据界中的大象，那么 Mahout 就是能让这头体态庞大的大象轻盈舞蹈的驱象人。

2. MLBase

MLBase 是由加利福尼亚大学伯克利分校 AMP 实验室推出的一个基于 Spark 的分布式数据挖掘解决方案，它是 Spark 生态圈的一部分，专门负责机器学习（除此之外，还有负责图计算的 GraphX、SQL ad-hoc 查询的 Shark、具备容错性查询能力的 BlinkDB 等）。AMP 实验室在 2014 年公开发表了关于 MLBase 的论文，AMP 实验室表示会开源整个项目。与类似的解决方案相比，MLBase 的构想有更进一步的创新和独到之处。比如相对于 Weka，MLBase 提供了分布式的数据挖掘解决方案，而 Weka 只支持单机；而相对于基于 Hadoop 的 Mahout 而言，MLBase 能更好地支持迭代计算，更重要的是 MLBase 的核心部分包括了一个优化器 ML Optimizer，它把数据拆成若干份，对每一份使用不同的算法和参数来运算出结果，看哪一种搭配方式得到的结果最优（注意这个最优结果是初步的）。MLBase 一共包括 4 个部分：ML Optimizer、MLI、MLlib 和 Spark。从 MLlib、MLI 到 ML Optimizer，是针对不同程度的算法科学家使用的不同程度的接口。MLBase 架构如图 10.8 所示。

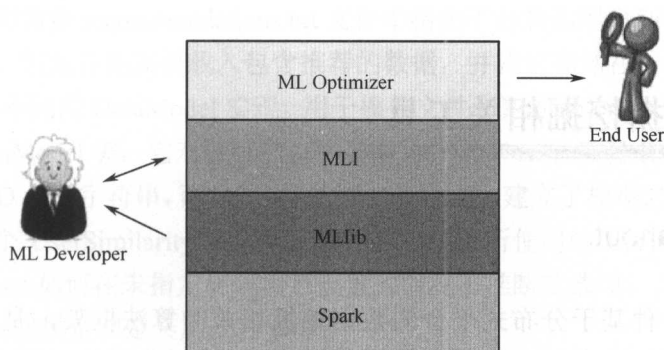


图 10.8 MLBase 架构

MLBase 设计架构不仅考虑到分布式的数据挖掘，还考虑到让数据挖掘的门槛更低，让一些可能并不了解数据挖掘的用户也能使用 MLBase 这个工具来处理自己的数据。那么 MLBase 是怎么做到这些的呢？首先，MLBase 提供了一套声明式的类 Pig 的语言。比如要

做分类，用户只需要写以下几行 Scala 代码：

```
var X = load("raw_data", 2 to 10)
var Y = load("raw_data", 1)
var (fn-model, summary) = doClassify(X, Y)
```

上面三行代码表示 X 是需要分类的数据集， Y 是从这个数据集里取的一个分类标签，用 `doClassify()` 做分类。这样的处理有两个主要好处：第一，每一步数据处理十分清楚简单，可以很容易地可视化出来；第二，对用户来说，用 ML 算法处理这件事非常透明。那么 MLBase 是怎么做的呢？整个透明过程的逻辑如下。

用户输入类 Pig 的任务，比如做分类 `doClassify(X, Y)`，或者做协同过滤 `doCollabFilter(X, Y)`，还可以做一些图计算之类的任务。这些任务首先会经过解析器处理，然后交给逻辑学习计划（LLP）组件。LLP（Logical Learning Plan），是逻辑上的一个学习选择过程，在这个过程中完成 ML 算法选择、特征提取方法选择和参数选择。LLP 完成之后，交给优化器。优化器是 MLBase 的核心，它会把数据拆分成若干份，对每一份使用不同的算法和参数运算出结果，看哪一种搭配方式得到的结果最优。优化器做完这些事之后就交给物理学习计划（Physical Learning Plan, PLP）组件。PLP 会执行之前选好的算法方案，把结果计算出来并返回，同时返回这次计算的学习模型。总而言之，这个流程是 Task → Parser → LLP → Optimizer → PLP → Execute → Result/Model，即先从逻辑上，在已有的算法里选几个适合这个场景的组合，让优化器都去做一遍，再把最优的组合交给实际执行的部分去执行，返回结果。

除此之外，LLP 内部实现的算法是可以扩充的，MLBase 考虑到了可扩展性，就是想让 ML 专家增加新的 ML 算法到 MLBase 里去，这样可以基于众包的概念来发展和壮大其算法库。

10.3 数据挖掘相关工具

10.3.1 Mahout

Mahout 是一种基于分布式平台的数据挖掘相关的算法框架，是 Apache Software Foundation (ASF) 旗下的一个开源项目，提供一些可扩展的机器学习领域经典算法的实现，旨在帮助开发人员更加方便快捷地创建智能应用程序。Apache Mahout 项目已经发展到了它的第三个年头，目前已经有了三个公共发行版本。Mahout 包含许多实现，包括聚类、分类、推荐过滤。此外，通过使用 Apache Hadoop 库，Mahout 可以有效地扩展到云中。

Mahout 主要实现了推荐系统、聚类和分类三大类数据挖掘算法。

1. Mahout 推荐系统模块

Mahout 目前提供了一些工具，可用于通过 Taste 库建立一个推荐引擎——针对 CF 的快速且灵活的引擎。Taste 支持基于用户和基于项目的推荐，并且提供了许多推荐选项，以及用于自定义的界面。Taste 包含 5 个主要组件，用于操作用户、项目和首选项。

- ① DataModel: 用于存储用户、项目和首选项。
- ② UserSimilarity: 用于定义两个用户之间的相似度的界面。
- ③ ItemSimilarity: 用于定义两个项目之间的相似度的界面。
- ④ Recommender: 用于提供推荐的界面。
- ⑤ UserNeighborhood: 用于计算相似用户邻近度的界面，其结果随时可由 Recommender 使用。

借助这些组件以及它们的实现，开发人员可以构建复杂的推荐系统，提供基于实时或者离线的推荐。基于实时的推荐经常只能处理数千名用户，而离线推荐具有更好的适用性。Taste 甚至提供了一些可利用 Hadoop 离线计算推荐的工具。在许多情况下，这种合适的方法可以帮助满足包含大量用户、项目和首选项的大型系统的需求。

为了演示如何构建一个简单的推荐系统，需要一些用户、项目和评分。为此，本节会使用 `cf.wikipedia.GenerateRatings` 中的代码（包含在示例代码的源代码中）为 Wikipedia 文档（Taste 称之为项目）随机生成大量用户和首选项，然后手动补充一些关于特定话题（Abraham Lincoln）的评分，从而创建示例中的最终文件 `recommendations.txt`。此方法的内涵是展示 CF 如何将对某特定话题感兴趣的人导向相关话题的其他文档。此示例的数据来源于 990 个随机用户（标记为 0~989），他们为集合中的所有文章随机分配了一些评分；以及 10 个用户（标记为 990~999），他们对集合中包含关键字 Abraham Lincoln 的 17 篇文章中的部分文章进行了评分。

首先演示如何为在 `recommendations.txt` 文件中指定了分数的用户创建推荐。这是 Taste 最为常见的应用，因此首先需要载入包含推荐的数据，并将它存储在一个 DataModel 中。Taste 提供了一些不同的 DataModel 实现，用于操作文件和数据库。在本例中，为简便起见，选择使用 FileDataModel 类，它对各行的格式要求为：用户 ID、项目 ID、首选项。其中，用户 ID 和项目 ID 都是字符串，而首选项可以是双精度型。建立了模型之后，需要通知 Taste 如何通过声明一个 UserSimilarity 实现来比较用户。根据所使用的 UserSimilarity 实现，可能还需要通知 Taste 如何在未指定明确用户设置的情况下推断首选项。

清单 1——创建模型和定义用户相似度：

```
//create the data model
FileDataModel dataModel = new FileDataModel(new File(recsFile));
UserSimilarity userSimilarity = new PearsonCorrelationSimilarity(dataModel);
// Optional:
userSimilarity.setPreferenceInfererrer(new AveragingPreferenceInfererrer(dataModel));
```


在清单 1 中，使用了 `PearsonCorrelationSimilarity`，它用于度量两个变量之间的关系，也可以使用其他 `UserSimilarity` 度量。应该根据数据和测试类型来选择相似度度量。

为了完成此示例，需要构建一个 `UserNeighborhood` 和一个 `Recommender`。`UserNeighborhood` 可以识别与相关用户类似的用户，并传递给 `Recommender`，后者将负责创建推荐项目排名表。

清单 2——生成推荐：

```
//Get a neighborhood of users
UserNeighborhood neighborhood = new NearestUserNeighborhood(neighborhoodSize,
userSimilarity, dataModel);

//Create the recommender
Recommender recommender= new GenericUserBasedRecommender(
dataModel, neighborhood, userSimilarity);
User user = dataModel.getUser(userId);
System.out.println("-----");
System.out.println("User: " + user);
//Print out the users own preferences first
TasteUtils.printPreferences(user, handler.map);
//Get the top 5 recommendations
List<RecommendedItem> recommendations = recommender.recommend(userId, 5);
TasteUtils.printRecs(recommendations, handler.map);
```

可以在命令行中运行整个示例，方法是在包含示例的目录中执行 `ant user-demo` 命令。执行此命令将打印输出虚构用户 995 的首选项和推荐，该用户只是 Lincoln 的爱好者之一。清单 3 显示了执行 `ant user-demo` 命令的输出。

清单 3——用户推荐的输出：

```
[echo] Getting similar items for user: 995 with a neighborhood of 5
[java] 09/08/20 08:13:51 INFO file.FileDataModel: Creating FileDataModel
        for file src/main/resources/recommendations.txt
[java] 09/08/20 08:13:51 INFO file.FileDataModel: Reading file info...
[java] 09/08/20 08:13:51 INFO file.FileDataModel: Processed 100000 lines
[java] 09/08/20 08:13:51 INFO file.FileDataModel: Read lines: 111901
[java] Data Model: Users: 1000 Items: 2284
[java] -----
[java] User: 995
[java] Title: August 21 Rating: 3.930000066757202
[java] Title: April Rating: 2.203000068664551
```



```

[java] Title: April 11 Rating: 4.230000019073486
[java] Title: Battle of Gettysburg Rating: 5.0
[java] Title: Abraham Lincoln Rating: 4.739999771118164
[java] Title: History of The Church of Jesus Christ of Latter-day Saints
        Rating: 3.430000066757202
[java] Title: Boston Corbett Rating: 2.009999990463257
[java] Title: Atlanta, Georgia Rating: 4.429999828338623
[java] Recommendations:
[java] Doc Id: 50575 Title: April 10 Score: 4.98
[java] Doc Id: 134101348 Title: April 26 Score: 4.860541
[java] Doc Id: 133445748 Title: Folklore of the United States Score: 4.4308662
[java] Doc Id: 1193764 Title: Brigham Young Score: 4.404066
[java] Doc Id: 2417937 Title: Andrew Johnson Score: 4.24178

```

从清单 3 中可以看到，系统推荐了一些可信级别不同的文章。事实上，这些项目的分数都是由其他 Lincoln 爱好者指定的，而不是用户 995 一人所为。如果希望查看其他用户的结构，只需要在命令行中传递 `-Duser.id=USER-ID` 参数，其中 `USER-ID` 是 0 和 999 之间的编号。还可以通过传递 `-Dneighbor.size=X` 来更改邻近空间，其中 `X` 是一个大于 0 的整型值。事实上，将邻近空间更改为 10 可以生成极为不同的结果，这是因为邻近范围内存在一个随机用户。要查看邻近用户以及共有的项目，可以向命令行添加 `-Dcommon=true`。

如前所述，基于用户的方法经常不具有可伸缩性。在本例中，使用基于项目的方法是更好的选择。幸运的是，Taste 可以非常轻松地实现基于项目的方法。处理项目相似度的基本代码并没有很大差异。

清单 4——项目相似度示例（摘录自 `cf.wikipedia.WikipediaTasteItemItemDemo`）：

```

//create the data model
FileDataModel dataModel = new FileDataModel(new File(recsFile));
//Create an ItemSimilarity
ItemSimilarity itemSimilarity = new LogLikelihoodSimilarity(dataModel);
//Create an Item Based Recommender
ItemBasedRecommender recommender =
    new GenericItemBasedRecommender(dataModel, itemSimilarity);
//Get the recommendations
List<RecommendedItem> recommendations =
    recommender.recommend(userId, 5);
TasteUtils.printRecs(recommendations, handler.map);

```

与清单 1 相同，清单 4 中根据推荐文件创建了一个 `DataModel`，但这次并未实例化 `UserSimilarity` 实例，而是使用 `LogLikelihoodSimilarity` 创建了一个 `ItemSimilarity`，它可以帮助处理不常见的事件。然后，将 `ItemSimilarity` 提供给 `ItemBasedRecommender`，最后请求推荐。当然，在此基础上，可以让系统支持离线执行这些计算，还可以探索其他的 `ItemSimilarity` 度量。注意，由于本例中的数据是随机的，所推荐的内容可能并不符合用户的期望。

再来看新用户的例子，当用户导航到某个项目之后，缺少用户首选项时的操作就比较容易实现了。对于这种情况，可以利用项目计算并向 `ItemBasedRecommender` 请求与当前项目最相似的项目。

清单 5——相似项目演示（摘录自 `cf.wikipedia.WikipediaTasteItemRecDemo`）：

```
//create the data model
FileDataModel dataModel = new FileDataModel(new File(recsFile));

//Create an ItemSimilarity
ItemSimilarity itemSimilarity = new LogLikelihoodSimilarity(dataModel);

//Create an Item Based Recommender
ItemBasedRecommender recommender =
    new GenericItemBasedRecommender(dataModel, itemSimilarity);

//Get the recommendations for the Item
List<RecommendedItem> simItems
    = recommender.mostSimilarItems(itemId, numRecs);

TasteUtils.printRecs(simItems, handler.map);
```

可以在命令行中执行 `ant sim-item-demo` 命令来运行清单 5。它与清单 4 之间的唯一区别就是，清单 5 并没有请求推荐，而是请求输出最相似的项目。

2. Mahout 聚类模块

Mahout 支持一些聚类算法实现（都是使用 `MapReduce` 编写的），它们都有各自的目标和标准。

① **Canopy**：一种快速聚类算法，通常用于为其他聚类算法创建初始种子。

② **k-Means**（以及模糊 *k*-Means）：根据项目与之前迭代的质心（或中心）之间的距离将项目添加到 *k* 簇中。

③ **Mean-Shift**：不需要任何关于聚类数量的推理知识的算法，它可以生成任意形状的簇。

④ **Dirichlet**：借助基于多种概率模型的聚类，它不需要提前执行特定的聚类视图。使用 Mahout 创建数据聚类的步骤如下。

① 准备输入。如果创建文本簇，则需要将文本转换成数值表示。

- ② 使用 Mahout 中可用的 Hadoop 就绪的驱动程序运行所选聚类算法。
- ③ 计算结果。
- ④ 如果有必要，则执行迭代。

聚类算法要求数据必须采用适合处理的格式。在机器学习中，数据通常被表示为矢量，有时也称为特征矢量。Mahout 提供了两个 Vector 表示：DenseVector 和 SparseVector。通常而言，基于文本的问题是很少的，因此应该使用 SparseVector 来处理文本。另一方面，如果大多数矢量的大多数值都是非零的，则比较适合使用 DenseVector。

通过 Wikipedia 内容生成矢量的方法如下。

将内容索引编入 Lucene，确存储相关字段(用于生成矢量的字段)的 term 矢量。Lucene 提供了一个名为 EnWikiDocMaker 的类(包含在 Lucene 的 contrib/benchmark 包中)，该类可以读取 Wikipedia 文件块中的内容并生成编入 Lucene 索引的文档。使用 org.apache.mahout.utils.vectors.lucene.Driver 类(位于 Mahout 的 utils 模块中)通过 Lucene 索引创建矢量。

创建了一组矢量之后，接下来需要运行 k-Means 集群算法。Mahout 为所有集群算法都提供了驱动程序，包括 k-Means 算法(KMeansDriver)。可以直接将驱动程序作为单独的程使用，而不需要 Hadoop 的支持，比如可以直接运行 ant k-means 命令。完成此操作之后，可以使用 ant dump 命令打印输出结果。

成功在独立模式中运行驱动程序之后，可以继续使用 Hadoop 的分布式模式。为此，需要 Mahout Job JAR，它位于示例代码的 hadoop 目录中。Job JAR 包可以将所有代码和依赖关系打包到一个 JAR 文件中，以便于加载到 Hadoop 中。

3. Mahout 分类模块

Mahout 目前支持两种根据贝氏统计来实现内容分类的方法。第一种方法是使用简单的支持 MapReduce 的 Naive Bayes 分类器。Naive Bayes 分类器以速度快和准确性高而著称，但其关于数据的简单(通常也是不正确的)假设是完全独立的。当各类训练示例的大小不平衡，或者数据的独立性不符合要求时，Naive Bayes 分类器会出现故障。第二种方法是 Complementary Naive Bayes，它会尝试纠正 Naive Bayes 方法中的一些问题，同时仍然能够维持简单性和速度。

简单来讲，Naive Bayes 分类器包括两个流程：跟踪特定文档及类别相关的特征(词汇)，然后使用此信息预测新的、未见过的内容的类别。第一个步骤称为训练(training)，它将通过查看已分类内容的示例来创建一个模型，然后跟踪与特定内容相关的各个词汇的概率。第二个步骤称为分类，它将使用在训练阶段中创建的模型以及新文档的内容，并结合 Bayes Theorem 来预测传入文档的类别。因此，要运行 Mahout 的分类器，首先需要训练模式，然后使用该模式对新内容进行分类。

10.3.2 语言工具——Python

在算法选型和数据验证阶段，出于效率考虑我们并不需要直接在分布式平台上进行试验。这时，高效的数据挖掘单机平台就显得十分重要。

选择 Python 作为数据挖掘单机平台主要出于如下三个原因：①Python 的语法清晰；②易于操作纯文本文件；③使用广泛，存在大量开发文档。目前越来越多的数据挖掘项目用 Python 来开发，Python 相关的库越来越丰富庞大。

1. Python 数据挖掘的优势和劣势

Python 的第一个优势是它是一种可执行的伪码。可执行的伪码是指 Python 的语法十分清晰可读。默认的 Python 开发环境已经集成了许多高级数据类型（列表、元组、字典、集合、队列等）。这些集合使得编程人员在实现数学概念时十分方便。此外 Python 还支持诸如面向对象编程、面向过程编程以及函数式编程等诸多编程范式。

Python 的第二个优势是它处理和操作文本文件非常简单，且非常善于处理数值型数据。在数据挖掘过程中，源数据很可能是以文本方式提供的，Python 提供的诸如强大的正则表达式函数以及 Web 数据提取接口，都大大提高了处理源数据的效率。

Python 的第三个优势在于它是一种广泛流行的语言。广泛流行意味着 Python 的代码示例很多，便于快速学习和掌握。广泛流行的另一个优势是，Python 存在大量的各个领域的模块库，可以大大缩短开发周期。

Python 的 SciPy 和 NumPy 模块为高效的机器学习算法实现提供了基础。Matplotlib 为 Python 提供了强大的绘图功能。

综上所述，Python 是一种支持矩阵运算的强大的免费数据挖掘工具。与 MATLAB 等专业工具相比，Python 有着不输于它们的性能、低廉的成本以及它们所不具备的活跃的开源社区支持。与 Java 和 C 等强类型编程语言相比（强类型语言实现一个简单的计算步骤都需要大量的代码，程序员往往被迫沉浸于代码细节中），Python 语言清晰简练，可以将程序员从代码中解放出来，专注于模型和数据。

当然，Python 语言也有很多缺点。最大的问题是 Python 的性能问题，这个问题可以通过调用 C 模块部分解决。可以先使用 Python 验证算法，然后在后续过程中逐步使用 C 代码替换 Python。

2. NumPy 矩阵运算库

NumPy 是 Python 的一种基础科学计算包。它的主要功能是为 Python 提供像 MATLAB 那样的基本向量和矩阵运算能力。这对实现数据挖掘算法非常重要，没有矩阵运算的使用，程序员就不得不编写大量的循环语句，这不仅会增加程序的复杂性，更会拖慢算法的运行速度。没有 NumPy 提供底层高效的矩阵运算支持，Python 就不可能成为一种数据挖掘平台。

NumPy 包含的功能如下:

- ① 一种强大的 N 维数组对象;
- ② 精心设计的矩阵运算函数;
- ③ 用来集成 C/C++ 和 FORTRAN 代码的工具;
- ④ 线性代数、傅里叶变换以及随机数生成工具。

除了科学计算的相关支持外, NumPy 还提供了高效的多维泛型数据容器。它可以定义任意类型的数据。这使得 NumPy 可以与大多数数据库无缝高速集成。

3. SciPy 科学计算库

SciPy 是一个开源的 Python 算法库和数学工具包。SciPy 包含的模块有最优化、线性代数、积分、插值、特殊函数、快速傅里叶变换、信号处理、图像处理、常微分方程求解和其他科学与工程中常用的计算。

在数据挖掘中, 经常使用的是其线性代数库, 如矩阵分解等。

4. Sci-kit:learn 机器学习库

Sci-kit:learn 是 Python 下一种最主要的开源机器学习库。它实现了数据挖掘的大部分方法, 算法主要构建在 SciPy、NumPy 和 Matplotlib 的平台基础之上, 因此运行效率很高。

Sci-kit: learn 目前主要支持的算法如下:

- ① 分类算法, 如 SVM、近邻算法、随机森林、决策树、LR 等;
- ② 回归算法, 如 SVR、岭回归、Lasso 等;
- ③ 聚类算法, 如 K 均值、谱聚类等。

此外它还支持其他数据挖掘必需的工具算法, 如:

- ① 降维, 包括 PCA、特征选择、矩阵分解等;
- ② 模型选择, 包括交叉验证、网格搜索等;
- ③ 数据预处理。

10.4 数据挖掘与 R 语言

10.4.1 R 语言简介

R 语言是主要用于统计分析、绘图的语言和操作环境。R 语言原来由来自新西兰奥克兰大学的 Ross Ihaka 和 Robert Gentleman 开发, 现在由“R 语言开发核心团队”负责开发。R 语言是基于 S 语言的一个 GNU 项目, 所以也可以把它当作 S 语言的一种实现, 通常用 S 语言编写的代码都可以不做修改地在 R 语言环境下运行。

R 语言的源代码可自由下载使用, 也有已编译的执行档版本可以下载, 可在多种平台

下运行，包括 UNIX（也包括 FreeBSD 和 Linux）、Windows 和 Mac OS。R 语言主要以命令行操作，同时有人开发了几种图形用户界面。

R 语言内建了多种统计学及数字分析功能。因为 S 语言的血缘，R 语言比其他统计学或数学专用的编程语言有更强的物件导向（面向对象程序设计）功能。R 语言的另一强项是绘图功能。

虽然 R 语言主要用于统计分析或者开发统计相关的软件，但也有人将其用于矩阵计算。其分析速度可媲美 GNU Octave 甚至商业软件 MATLAB。

R 语言能够通过由用户编写的软件包增加功能。增加的功能有特殊的统计技术、绘图功能，以及编程界面和数据输入/输出功能。这些软件包是用 R 语言、LaTeX、Java 及最常用的 C 语言和 FORTRAN 编写的。

R 语言是一套完整的数据处理、计算和制图软件系统。其功能包括：数据存储和处理系统；数组运算工具（其向量、矩阵运算方面的功能尤其强大）；完整连贯的统计分析工具；优秀的统计制图功能；简便而强大的编程语言，可操纵数据的输入和输出，可实现分支、循环，用户可自定义功能。

与其说 R 语言是一种统计软件，还不如说 R 语言是一种数学计算环境。R 语言不仅提供一些集成的统计工具，而且提供各种数学计算、统计计算的函数，从而使使用者能灵活机动地进行数据分析，甚至创造出符合需要的新的统计计算方法。

该语言的语法表面上类似 C 语言，但在语义上是函数设计语言的变种，并且和 Lisp 以及 APL 有很强的兼容性。更为特别的是，它允许在“语言上计算”。这使得它可以把表达式作为函数的输入参数，而这种做法对统计模拟和绘图非常有用。

10.4.2 R 语言在数据挖掘上的应用

本节介绍用 R 语言对 Iris 数据集进行聚类分析。

1. 第一步：对数据集进行初步统计分析

检查数据的维度：

```
> dim(iris)
[1] 150  5
```

显示数据集中的列名：

```
> names(iris)
[1] "Sepal.Length" "Sepal.Width" "Petal.Length" "Petal.Width" "Species"
```

显示数据集的内部结构：

```
> str(iris)
'data.frame': 150 obs. of 5 variables:
 $ Sepal.Length: num 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
```

```
$ Sepal.Width : num 3.5 3.3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
$ Petal.Length: num 1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
$ Petal.Width : num 0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
$ Species      : Factor w/ 3 levels "setosa","versicolor",...: 1 1 1 1 1 1 1 1 1 1 ...
```

显示数据集的属性:

```
> attributes(iris)
$names
[1] "Sepal.Length" "Sepal.Width"  "Petal.Length" "Petal.Width"  "Species"
$row.names
[1] 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20
[21] 21 22 23 24 25 26 27 28 29 30 31 32 33 34 35 36 37 38 39 40
[41] 41 42 43 44 45 46 47 48 49 50 51 52 53 54 55 56 57 58 59 60
[61] 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75 76 77 78 79 80
[81] 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99 100
[101] 101 102 103 104 105 106 107 108 109 110 111 112 113 114 115 116 117 118 119 120
[121] 121 122 123 124 125 126 127 128 129 130 131 132 133 134 135 136 137 138 139 140
[141] 141 142 143 144 145 146 147 148 149 150
$class
[1] "data.frame"
```

查看数据集的前 5 项数据情况:

```
> iris[1:5,]
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1           5.1          3.5           1.4          0.2  setosa
2           4.9          3.0           1.4          0.2  setosa
3           4.7          3.2           1.3          0.2  setosa
4           4.6          3.1           1.5          0.2  setosa
5           5.0          3.6           1.4          0.2  setosa
```

查看数据集中属性 Sepal.Length 前 10 行数据:

```
> iris[1:10, "Sepal.Length"]
[1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9
```

目的同上:

```
> iris$Sepal.Length[1:10]
[1] 5.1 4.9 4.7 4.6 5.0 5.4 4.6 5.0 4.4 4.9
```

显示数据集中每个变量的分布情况：

```
> summary(iris)

Sepal.Length    Sepal.Width    Petal.Length    Petal.Width      Species
Min.   :4.300    Min.   :2.000    Min.   :1.000    Min.   :0.100    setosa   :50
1st Qu.:5.100    1st Qu.:2.800    1st Qu.:1.600    1st Qu.:0.300    versicolor:50
Median :5.800    Median :3.000    Median :4.350    Median :1.300    virginica :50
Mean   :5.843    Mean   :3.057    Mean   :3.758    Mean   :1.199
3rd Qu.:6.400    3rd Qu.:3.300    3rd Qu.:5.100    3rd Qu.:1.800
Max.   :7.900    Max.   :4.400    Max.   :6.900    Max.   :2.500
```

显示 Iris 数据集列 Species 中各个值出现的频次：

```
> table(iris$Species)

setosa versicolor virginica
     50       50       50
```

根据列 Species 画出饼图：

```
> pie(table(iris$Species))
```

算出列 Sepal.Length 中所有值的方差：

```
> var(iris$Sepal.Length)
[1] 0.6856935
```

算出列 iris\$Sepal.Length 和 iris\$Petal.Length 的协方差：

```
> cov(iris$Sepal.Length, iris$Petal.Length)
[1] 1.274315
```

算出列 iris\$Sepal.Length 和 iris\$Petal.Length 的相关系数，从结果看这两个值是强相关的：

```
> cor(iris$Sepal.Length, iris$Petal.Length)
[1] 0.8717538
```

画出列 iris\$Sepal.Length 的分布柱状图：

```
> hist(iris$Sepal.Length)
```

画出列 iris\$Sepal.Length 的密度函数图：

```
> plot(density(iris$Sepal.Length))
```

画出列 iris\$Sepal.Length 和 iris\$Sepal.Width 的散点图：

```
> plot(iris$Sepal.Length, iris$Sepal.Width)
```

绘出矩阵各列的散布图：

```
> plot(iris)
or
> pairs(iris)
```


totss 表示所生成聚类的总体距离平方和。

withinss 表示各个聚类组内的距离平方和。

tot.withinss 表示聚类组内的距离平方和总量。

betweenss 表示聚类组间的距离平方和总量。

size 表示每个聚类组中成员的数量。

创建一个连续表，在三个聚类中分别统计各种花出现的次数：

```
> table(iris$Species, kc$cluster)
```

```
      1  2  3
setosa  0 50  0
versicolo
r      2  0 48
virginica 36  0 14
```

根据最后的聚类结果画出散点图，数据为结果集中的列 Sepal.Length 和 Sepal.Width，颜色为用 1、2、3 表示的默认颜色：

```
> plot(newiris[,c("Sepal.Length", "Sepal.Width")], col = kc$cluster)
```

在图上标出每个聚类的中心点：

```
> points(kc$centers[,c("Sepal.Length", "Sepal.Width")], col = 1:3, pch = 8, cex=2)
```

10.5 练习题

1. 常用的分类算法有哪些？
2. 了解 Mahout 中实现的分布式数据挖掘算法，并运行一个 K-means 实例。
3. 调研一下 R 语言和 Hadoop 的集成方法。

参考文献

- [1] uilan JR. Induction of decision trees. Machine Learning.
- [2] reiman L, Friedman J, Stone C. Classification and Regression Trees.
- [3] insky ML, Papert SA. Perceptrons. Cambridge, MA: MIT Press.
- [4] allant SI. Perceptron-based learning algorithms. IEEE Transactions on Neural Networks.
- [5] Cover T, Hart P. Nearest neighbor pattern classification. IEEE Transactions on Information Theory.

- [6] Hastie T, Tibshirani R, Friedman J. The Elements of Statistical Learning: Data Mining, Inference and prediction.
- [7] Bishop C. Pattern Recognition and Machine Learning.
- [8] Mitchell TM. Machine Learning.
- [9] Cortes C, Vapnik V. Support-vector networks.
- [10] Platt JC. Fast training of support vector machines using sequential minimal optimization.
- [11] K. Ali, S. Manganaris, and R. Srikant. Partial Classification using Association Rules.
- [12] R. Agrawal, T. Imielinski, and A. Swami. Mining association rules between sets of items in large databases.
- [13] B. Dunknel and N. Sopaekar. Data Organization and Access for Efficient Data Mining.
- [14] A.K.Jain and R.C.Dubes. Algorithms for Clustering Data.
- [15] A.K.Jain, M.N.Murty, and P.J Flynn. Data Clustering:A review.
- [16] L.Kaufman and P.J.Rousseeuw. Finding Groups in Data: An Introduction to Cluster Analysis.
- [17] <http://mahout.apache.org>
- [18] <http://www.scipy.org>
- [19] <http://scikit-learn.org/stable/index.html>
- [20] http://deeplearning.stanford.edu/wiki/index.php/Main_Page
- [21] <http://deeplearning.net/>
- [22] Yoshua Bengio, Ian Goodfellow, Aaron Courville. Deep Learning. MIT Press, In preparation. <http://www.iro.umontreal.ca/~bengioy/dlbook/>
- [23] Yoshua Bengio, Aaron Courville, Pascal Vincent. Representation Learning: A Review and New Perspectives. Arxiv, 2012.
- [24] The monograph or review paper Learning Deep Architectures for AI. Foundations & Trends in Machine Learning, 2009.
- [25] Itamar Arel, Derek C. Rose, and Thomas P. Karnowski. Deep Machine Learning – A New Frontier in Artificial Intelligence Research.
- [26] Graves, A. Supervised sequence labelling with recurrent neural networks(Vol. 385). Springer,2012.
- [27] Antoine Bordes, Xavier Glorot, Jason Weston and Yoshua Bengio. Joint Learning of Words and Meaning Representations for Open-Text Semantic Parsing. in: Proceedings of the 15th International Conference on Artificial Intelligence and Statistics (AISTATS) ,2012.
- [28] Socher, R., Huang, E. H., Pennington, J., Ng, A. Y., and Manning, C. D. Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. In NIPS'2011.
- [29] Socher, R., Pennington, J., Huang, E. H., Ng, A. Y., and Manning, C. D. Semi-supervised recursive autoencoders for predicting sentiment distributions. In EMNLP'2011.

- [30] Mikolov Tomáš. Statistical Language Models based on Neural Networks. PhD thesis, Brno University of Technology, 2012.
- [31] Graves, Alex, and Jürgen Schmidhuber. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Networks*, 2005 (18.5): 602-610.
- [32] Mikolov, Tomas, Ilya Sutskever, Kai Chen, Greg S. Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, 2013: 3111-3119.
- [33] Aaron Courville, James Bergstra and Yoshua Bengio. Unsupervised Models of Images by Spike-and-Slab RBMs. in: *ICML'2011*.
- [34] Hinton, Geoffrey. A practical guide to training restricted Boltzmann machines. *Momentum*, 2010 (9.1): 926.
- [35] Guillaume Alain, Yoshua Bengio and Salah Rifai. Regularized Auto-Encoders Estimate Local Statistics. Université de Montréal, arXiv report 1211.4246, 2012.
- [36] Salah Rifai, Yoshua Bengio, Yann Dauphin and Pascal Vincent. A Generative Process for Sampling Contractive Auto-Encoders. in: *ICML'2012*, Edinburgh, Scotland, U.K., 2012.
- [37] Salah Rifai, Pascal Vincent, Xavier Muller, Xavier Glorot and Yoshua Bengio. Contracting Auto-Encoders: Explicit invariance during feature extraction. in: *ICML'2011*.
- [38] Salah Rifai, Yoshua Bengio, Aaron Courville, Pascal Vincent and Mehdi Mirza. Disentangling factors of variation for facial expression recognition. in: *ECCV'2012*.
- [39] Vincent, Pascal, et al. Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion. *The Journal of Machine Learning Research*, 2010 (11): 3371-3408.
- [40] Vincent, Pascal. A connection between score matching and denoising autoencoders. *Neural computation*, 2011 (23.7): 1661-1674.
- [41] Chen, Minmin, et al. Marginalized denoising autoencoders for domain adaptation. arXiv preprint arXiv:1206.4683 (2012).
- [42] Salah Rifai, Yann Dauphin, Pascal Vincent, Yoshua Bengio and Xavier Muller. The Manifold Tangent Classifier. in: *NIPS'2011*.
- [43] Gens, Robert, and Pedro Domingos. Discriminative Learning of Sum-Product Networks. *NIPS 2012 Best Student Paper*.
- [44] Goodfellow, I., Warde-Farley, D., Mirza, M., Courville, A., and Bengio, Y. Maxout networks. Technical Report, Université de Montréal, 2013.
- [45] Hinton, Geoffrey E., et al. Improving neural networks by preventing co-adaptation of feature detectors. arXiv preprint arXiv:1207.0580 (2012).
- [46] Wang, Sida, and Christopher Manning. Fast dropout training. In *Proceedings of the 30th*

- International Conference on Machine Learning (ICML-13), 2013: 118-126.
- [47] Glorot, Xavier, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier networks. In Proceedings of the 14th International Conference on Artificial Intelligence and Statistics. JMLR W&CP Volume, 2011(15):315-323.
- [48] Alex Krizhevsky, Ilya Sutskever, Geoffrey E Hinton. Image Net Classification with Deep Convolutional Neural Networks. NIPS 2012.
- [49] Hinton, Geoffrey E. Deterministic Boltzmann learning performs steepest descent in weight-space. *Neural computation*, 1989 (1.1): 143-150.
- [50] Bengio, Yoshua, and Samy Bengio. Modeling high-dimensional discrete data with multi-layer neural networks. *Advances in Neural Information Processing Systems*, 2000(12): 400-406.
- [51] Bengio, Yoshua, et al. Greedy layer-wise training of deep networks. *Advances in neural information processing systems*, 2007 (19): 153.
- [52] Bengio, Yoshua, Martin Monperrus, and Hugo Larochelle. Nonlocal estimation of manifold structure. *Neural Computation*, 2006 (18.10): 2509-2528.
- [53] Hinton, Geoffrey E., and Ruslan R. Salakhutdinov. Reducing the dimensionality of data with neural networks. *Science*, 2006 (313.5786): 504-507.
- [54] Marc'Aurelio Ranzato, Y. Lan Boureau, and Yann LeCun. Sparse feature learning for deep belief networks. *Advances in neural information processing systems*, 2007 (20): 1185-1192.
- [55] Bengio, Yoshua, and Yann LeCun. Scaling learning algorithms towards AI. *Large-Scale Kernel Machines*, 2007 (34).
- [56] Le Roux, Nicolas, and Yoshua Bengio. Representational power of restricted boltzmann machines and deep belief networks. *Neural Computation*, 2008 (20.6): 1631-1649.
- [57] Sutskever, Ilya, and Geoffrey Hinton. Temporal-Kernel Recurrent Neural Networks. *Neural Networks*, 2010 (23.2): 239-243.
- [58] Le Roux, Nicolas, and Yoshua Bengio. Deep belief networks are compact universal approximators. *Neural computation*, 2010 (22.8): 2192-2207.
- [59] Bengio, Yoshua, and Olivier Delalleau. On the expressive power of deep architectures. *Algorithmic Learning Theory*. Springer Berlin/Heidelberg, 2011.
- [60] Montufar, Guido F., and Jason Morton. When Does a Mixture of Products Contain a Product of Mixtures?. *arXiv preprint arXiv:1206.0387* (2012).

第11章

深度学习

深度学习 (Deep Learning) 的概念由 Hinton 等人于 2006 年提出, 源于人工神经网络的研究, 是机器学习中一个非常接近人工智能的领域, 其动机在于建立模拟人脑进行分析学习的神经网络。深度学习是相对于简单学习而言的, 目前多数分类、回归等学习算法都属于简单学习, 其局限性在于有限样本和计算单元情况下对复杂函数的表示能力有限, 针对复杂分类问题其泛化能力受到一定制约。深度学习可通过学习一种深层非线性网络结构, 实现复杂函数逼近, 表征输入数据分布式表示, 其展现出了强大的从少数样本集中学习数据集本质特征的能力。深度学习模拟更多的神经层神经活动, 通过组合低层特征形成更加抽象的高层特征, 以发现数据的分布式特征表示, 深度学习的示意图如图 11.1 所示。

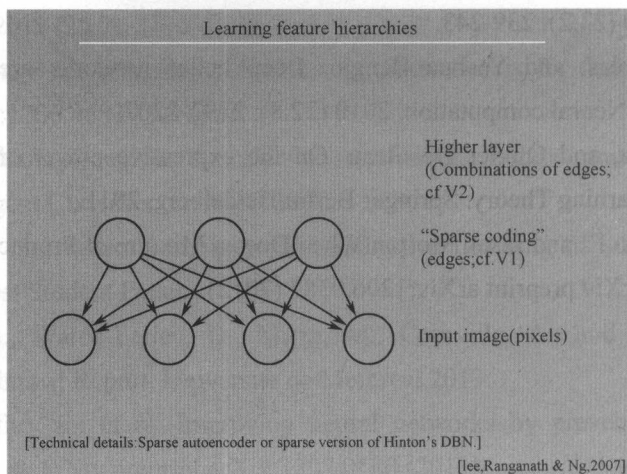


图 11.1 深度学习示意图

11.1 深度学习介绍

11.1.1 深度学习的概念

研究人员通过分析人脑的工作方式发现：通过感官信号从视网膜传递到前额大脑皮质再到运动神经的时间，推断出大脑皮质并未直接对数据进行特征提取处理，而是使接收到的刺激信号通过一个复杂的层状网络模型，进而获取观测数据展现的规则。也就是说，人脑并不是直接根据外部世界在视网膜上的投影来识别物体，而是根据经聚集和分解过程处理后的信息来识别物体。因此视皮层的功能是对感知信号进行特征提取和计算，而不仅仅是简单地重现视网膜的图像。人类感知系统这种明确的层次结构极大地降低了视觉系统处理的数据量，并保留了物体有用的结构信息。深度学习正是希望通过模拟人脑多层次的分析方式来提高学习的准确性。

实际生活中，人们为了解决一个问题，如对象的分类（对象可是文档、图像等），首先必须做的事情是表达一个对象，即必须抽取一些特征来表示一个对象，因此特征对结果的影响非常大。在传统的数据挖掘方法中，特征的选择一般都是通过手工完成的，通过手工选取的好处是可以借助人的经验或者专业知识选择出正确的特征；但缺点是效率低，而且在复杂的问题中，人工选择可能也会陷入困惑。于是，人们就在寻找一种能够自动选择特征，而且还能保证特征准确的方法。深度学习能够通过组合低层特征形成更抽象的高层特征，从而实现自动选择特征，而不需要人参与特征的选取。

接下来我们分析深度学习的核心思想。假设有一个系统 S ，它有 n 层 (S_1, \dots, S_n)，它的输入是 I ，输出是 O ，即 $I \rightarrow S_1 \rightarrow S_2 \rightarrow \dots \rightarrow S_n \rightarrow O$ ，如果输出 O 等于输入 I ，即输入 I 经过这个系统变化之后没有任何的信息损失，保持不变，则意味着输入 I 经过每一层 S_i 都没有任何的信息损失，即在任何一层 S_i ，它都是原有信息（即输入 I ）的另外一种表示。现在回到我们的主题深度学习中，我们需要自动地学习特征，假设有一堆输入 I （如一堆图像或者文本），并且设计了一个系统 S （有 n 层），通过调整系统中的参数，使得它的输出仍然是输入 I ，那么就可以自动获取输入 I 的一系列层次特征，即 S_1, \dots, S_n 。

对于深度学习来说，其思想就是堆叠多个层，也就是说上一层的输出作为下一层的输入。通过这种方式，就可以实现对输入信息进行分级表达。另外，之前假设输出严格地等于输入，这个限制过于严格，我们可以略微地放松这个限制，例如只要使得输入与输出的差别尽可能小即可，这个放松会引出另外一类不同的深度学习方法。

11.1.2 深度学习的结构

深度学习的结构有以下三种。

1. 生成性深度结构

生成性深度结构描述数据的高阶相关特性，或观测数据和相应类别的联合概率分布。与传统区分型神经网络不同，它可获取观测数据和标签的联合概率分布，这方便了先验概率和后验概率的估计，而区分型模型仅能对后验概率进行估计。论文“A fast learning algorithm for deep learning”中采用的深度置信网络（Deep Belief Network, DBN）就属于生成性深度结构。DBN 解决了传统 Back Propagation (BP) 算法训练多层神经网络的难题：①需要大量含标签训练样本集；②收敛速度较慢；③因不合适的参数选择而陷入局部最优。

DBN 由一系列受限玻尔兹曼机（Restricted Boltzmann Machine, RBM）单元组成。RBM 是一种典型神经网络，该网络可视层和隐层单元彼此互连（层内无连接），隐单元可获取输入可视单元的高阶相关性。相比于传统 Sigmoid 信度网络，RBM 权值的学习相对容易。为了获取生成性权值，预训练采用无监督贪心逐层方式来实现。在训练过程中，首先将可视向量值映射给隐单元，然后由隐单元重建可视单元，将这些新的可视单元再次映射给隐单元，就获取了新的隐单元。通过自底向上组合多个 RBM 可以构建一个 DBN。应用高斯—伯努利 RBM 或伯努利—伯努利 RBM，可用隐单元的输出作为训练上层伯努利—伯努利 RBM 的输入，第二层的输出作为第三层的输入等，如图 11.2 所示。

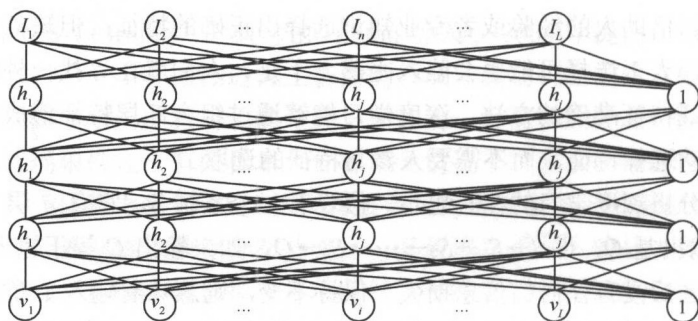


图 11.2 DBN 模型

2. 区分性深度结构

区分性深度结构的作用是提供对模式分类的区分性能力，通常描述数据的后验分布。卷积神经网络（Convolutional Neural Network, CNN）是第一个真正成功训练多层网络结构的学习算法，与 DBN 不同，它属于区分性训练算法。受视觉系统结构的启示，当具有相同参数的神经元应用于前一层的不同位置时，就可获取一种变换不变性特征。后来 LeCun 等人沿着这种思路，利用 BP 算法设计并训练了 CNN。CNN 作为深度学习框架是基于最小化预处理数据要求而产生的。受早期的时间延迟神经网络影响，CNN 靠共享时域权值降低复杂度。CNN 是利用空间关系减少参数数目以改善一般前向 BP 训练的一种拓扑结构，并在多个实验中获取了较好性能。在 CNN 中被称为局部感受区域的图像的一小部分作为分层结构的最底层输入。信息通过不同的网络层次进行传递，因此在每一层能够获取对平移、缩

放和旋转不变的观测数据的显著特征。

3. 混合型结构

混合型结构的学习过程包含两个部分，即生成性部分和区分性部分。现有典型的生成性单元通常最终用于区分性任务，生成性模型应用于分类任务时，预训练可结合其他典型区分性学习算法对所有权值进行优化。这个区分性寻优过程通常是附加一个顶层变量来表示训练集提供的期望输出或标签。BP 算法可用于优化 DBN 权值，它的初始权值在 RBM 和 DBN 预训练中得到而非随机产生，这样的网络通常会比仅通过 BP 算法单独训练的网络性能优越。可以认为 BP 对 DBN 训练仅完成局部参数空间搜索，与前馈型神经网络相比加速了训练和收敛。

11.1.3 从机器学习到深度学习

第 10 章中所述的机器学习算法无一例外要对数据集进行各种人工干预，具体来说，分为两个阶段的干预。首先，机器学习需要把数据表示成特征的集合，究竟用何种特征表示数据是由实现该算法的程序员决定的，这是第一阶段的人工干预，称为特征选择。第二阶段的人工干预产生于人们对于机器学习算法的选择，一旦选择了某种算法，就相当于假设数据集与这个算法的模型相似。

机器学习的终极目标是让计算机能够自己从数据中学习知识，从而为人服务。但是机器学习的人工干预使得这个目标无法实现，既然需要人工干预，就无法实现知识产生的自动化，不过是将人的想法用代码实现而已。但是在这些机器学习算法中，人工神经网络与其他算法有着明显的不同：①多层神经网络可以实现一种叫做自动编码器的算法，自动编码器的隐藏层实际上相当于一个自动的特征筛选过程，这个过程称为表示学习；②神经网络从理论上讲，与大多数机器学习算法相似，因此可以实现模型选择的自动化。

但是目前基于神经网络的特征表示基本都可以看成浅层学习，因为这些神经网络的隐藏层都很少。这是由于传统神经网络训练过程有很多局限性：①梯度扩散，传统算法在求解过程中依赖于后向传播的梯度信号，但是随着层数的增加，梯度误差校正信号的强度会逐渐变小，以至于最后不可用；②容易得到局部最优解，而非全局最优解；③对数据要求高，尤其是要求数据必须是有标签的数据，在实际中有标签的数据很难获得，而神经网络参数有很多，很可能无法训练出有效的模型。

虽然面对诸多困难，但浅层神经网络目前依然广泛应用于图像识别等领域，这说明少量的隐藏层在合理的调试下，依然能够被应用到现实中。但正因为调试困难，神经网络在数据挖掘中的应用没有其他机器学习算法广泛。

深度学习的出现，使得这个窘境有了解决的思路。深度学习的主要思想是增加神经网络中隐藏层的数量，使用大量的隐藏层来增强神经网络对特征筛选的能力，从而能够用较少的参数表达出复杂的模型函数，逼近机器学习的终极目标——知识的自动发现。

深度学习的核心技术就是一个能够有效解决传统神经训练方法种种问题的算法，这个算法将在后续部分中阐述。

11.2 深度学习基本方法

深度学习的训练方法如今已经有了许多复杂的变种实现，但这些实现的基本思想都是相同的，本节仅介绍最基本的深度学习算法的核心思想。

11.2.1 自动编码器

深度学习的基本算法被称为逐层贪心算法，该算法在每一次迭代中训练一层网络，然后使用一个类似于后向传播的算法对深度网络进行调优。具体来说，首先将深度网络看成一连串自动编码器。每个自动编码器可以看成由两个阶段构成，第一个阶段是编码阶段，编码阶段对应输入层到隐藏层的映射；第二个阶段是解码阶段，对应的是隐藏层到输出层的映射。自动编码器实现编码的学习过程如下：首先用隐藏层进行编码，再将编码结果作为输入传递给输出层进行解码，解码后的结果应该与原始输入相似但并不相同，通过将结果与原始输入的误差最小化得到最优编码方案，把中间层参数提取出来就是一个最优编码方案。

1. 前向训练阶段

在逐层贪心算法中，我们在整体上将自动编码器拆开，编码过程用下面的公式表示：

$$\begin{aligned} a^{(l)} &= f(z^{(l)}) \\ z^{(l+1)} &= W^{(l,1)} a^{(l)} + b^{(l,1)} \end{aligned}$$

解码过程用下面的公式表示：

$$\begin{aligned} a^{(n+1)} &= f(z^{(n+1)}) \\ z^{(n+l+1)} &= W^{(n-l,2)} a^{(n+l)} + b^{(n-l,2)} \end{aligned}$$

其中， $a^{(n)}$ 就包含了我们想要的高阶特征。

一个更具体的含两个隐藏层的训练步骤如下。

- ① 首先训练第一层自动编码器，如图 11.3 所示。
- ② 然后将第一层自动编码器的解码部分拿掉，直接将第一层的编码结果作为输入，利用这个输入训练第二层编码器（图 11.4）。
- ③ 最后根据需要将第二层的解码部分换成相应的分类函数即可实现一个简单的分类器（图 11.5）。

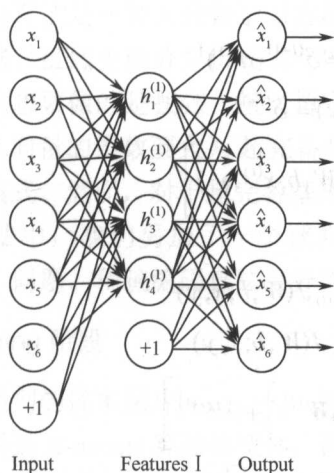


图 11.3 自动编码器训练步骤 1

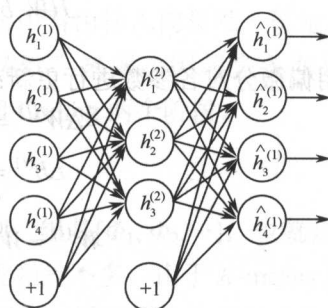


图 11.4 自动编码器训练步骤 2

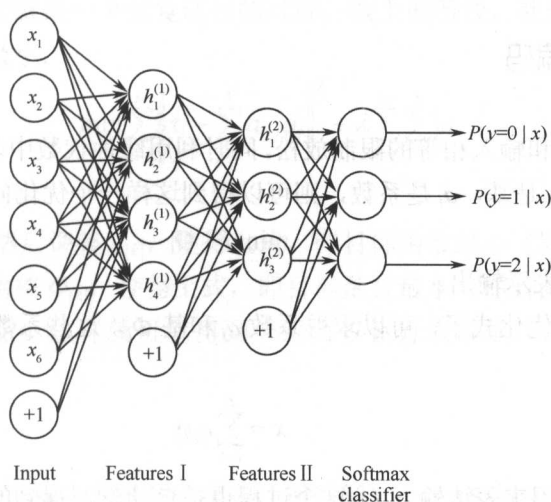


图 11.5 自动编码器训练步骤 3

2. 后向调优阶段

后向调优阶段的调优过程如下。

① 从输出层 n_l 开始，计算参数：

$$\delta^{(n_l)} = -(\nabla_{a^{(n_l)}} J) \cdot f'(z^{(n_l)})$$

② 对于 $l = n_l - 1, n_l - 2, \dots, 2$ 层，计算参数：

$$\delta^{(l)} = ((W^{(l+1)})^T \delta^{(l+1)}) \cdot f'(z^{(l)})$$

③ 计算目标偏微分:

$$\nabla_{W^{(l)}} J(W, b; x, y) = \delta^{(l+1)} (a^{(l)})^T$$

$$\nabla_{b^{(l)}} J(W, b; x, y) = \delta^{(l+1)}$$

$$J(W, b) = \left[\frac{1}{m} \sum_{i=1}^m J(W, b; x^{(i)}, y^{(i)}) \right]$$

④ 使用偏微分对各参数进行更新:

$$\Delta W^{(l)} = \Delta W^{(l)} + \nabla_{W^{(l)}} J(W, b; x, y)$$

$$\Delta b^{(l)} = \Delta b^{(l)} + \nabla_{b^{(l)}} J(W, b; x, y)$$

$$W^{(l)} = W^{(l)} - a \left[\left(\frac{1}{m} \Delta W^{(l)} \right) + \lambda W^{(l)} \right]$$

$$b^{(l)} = b^{(l)} - a \left[\left(\frac{1}{m} \Delta b^{(l)} \right) \right]$$

⑤ 完成更新后, 即完成一次优化迭代。

11.2.2 稀疏编码

如果把输出必须和输入相等的限制放松, 同时利用线性代数中基的概念, 即 $O = a_1 \times \Phi_1 + a_2 \times \Phi_2 + \dots + a_n \times \Phi_n$, Φ_i 是基, a_i 是系数, 则可以得到这样一个优化问题:

$$\min |I - O|$$

其中 I 表示输入, O 表示输出。

通过求解这个最优化式子, 可以求得系数 a_i 和基 Φ_i , 这些系数和基就是输入的另外一种近似表达。

$$x = \sum_{i=1}^k a_i \phi_i$$

因此, 它们可以用来表达输入 I , 这个过程也是自动学习得到的。如果在上述式子中加上 L_1 的正则因子限制, 则得到

$$\min |I - O| + u \times (|a_1| + |a_2| + \dots + |a_n|)$$

这种方法被称为稀疏编码 (Sparse Coding)。通俗地说, 就是将一个信号表示为一组基的线性组合, 而且要求只需要较少的几个基就可以将信号表示出来。稀疏性定义为: 只有很少的几个非零元素或只有很少的几个远大于零的元素。要求系数 a_i 稀疏的意思就是: 对于一组输入向量, 只想有尽可能少的几个系数远大于零。选择使用具有稀疏性的分量来表示输入数据是有原因的, 因为绝大多数的感官数据, 比如自然图像, 可以被表示成少量基本元素的叠加, 在图像中这些基本元素可以是面或者线。同时, 比如与初级视觉皮层的类比过程也因此得到了提升 (人脑有大量的神经元, 但对于某些图像或者边缘只有很少的神经元兴奋, 其他都处于抑制状态)。

稀疏编码算法是一种无监督学习方法，它用来寻找一组“超完备”基向量以更高效地表示样本数据。虽然主成分分析技术（PCA）能使我们方便地找到一组“完备”基向量，但是这里我们想要做的是找到一组“超完备”基向量来表示输入向量（也就是说，基向量的个数比输入向量的维数要大）。超完备基的好处是它们能更有效地找出隐含在输入数据内部的结构与模式。然而，对于超完备基来说，系数 a_i 不再由输入向量唯一确定。因此，在稀疏编码算法中，我们另加了一个评判标准“稀疏性”来解决因超完备而导致的退化（degeneracy）问题。稀疏编码算法可分为 Training 和 Coding 两个阶段。

1. Training 阶段

给定一系列的样本图片 $[x_1, x_2, \dots]$ ，需要学习得到一组基 $[\Phi_1, \Phi_2, \dots]$ ，也就是字典。

稀疏编码是 K -means 算法的变体，两者训练过程相差不多。由于 K -means 聚类算法为 EM 算法的具体应用，这里简单介绍 EM 算法的思想：如果要优化的目标函数包含两个变量，如 $L(W, B)$ ，那么可以先固定 W ，调整 B 使得 L 最小，然后再固定 B ，调整 W 使 L 最小，这样迭代交替，不断将 L 推向最小值。

稀疏编码的训练过程就是一个重复迭代的过程，按上面所说，我们交替地更改 a 和 Φ 使得下面这个目标函数最小：

$$\min_{a, \phi} \sum_{i=1}^m \left\| x_i - \sum_{j=1}^k a_{i,j} \phi_j \right\|^2 + \lambda \sum_{i=1}^m \sum_{j=1}^k |a_{i,j}|$$

每次迭代分两步：

① 固定字典 $\Phi_{[k]}$ ，然后调整 $a_{[k]}$ ，使得上式，即目标函数最小（即解 LASSO 问题）。

② 然后固定 $a_{[k]}$ ，调整 $\Phi_{[k]}$ ，使得上式，即目标函数最小（即解凸 QP 问题）。

不断迭代，直至收敛。这样就可以得到一组可以良好表示这一系列 x 的基，也就是字典。

2. Coding 阶段

给定一个新的图片 x ，由上面得到的字典，通过解一个 LASSO 问题得到稀疏向量 a 。这个稀疏向量就是这个输入向量 x 的一个稀疏表达。

$$\min_a \sum_{i=1}^m \left\| x_i - \sum_{j=1}^k a_{i,j} \phi_j \right\|^2 + \lambda \sum_{i=1}^m \sum_{j=1}^k |a_{i,j}|$$

11.3 深度学习模型

同机器学习方法一样，深度学习方法也有生成模型与判别模型之分，不同的学习框架下建立的学习模型不同。例如，卷积神经网络就是一种深度判别模型，而深度置信网络就是一种生成模型。

11.3.1 深度置信网络

深度置信网络（DBN）是一个概率生成模型，与传统的判别模型的神经网络相对，生成模型是建立一个观察数据和标签之间的联合分布，对 $P(\text{Observation}|\text{Label})$ 和 $P(\text{Label}|\text{Observation})$ 都做了评估；而判别模型仅仅评估了后者，也就是 $P(\text{Label}|\text{Observation})$ 。对深度神经网络应用传统的 BP 算法时，DBN 遇到了以下问题：

- ① 需要为训练提供一个有标签的样本集；
- ② 学习过程较慢；
- ③ 不适当的参数选择会导致学习收敛于局部最优解。

DBN 由多个受限玻尔兹曼机（Restricted Boltzmann Machines, RBM）层组成，一个典型的 DBN 结构如图 11.6 所示。这些网络被“限制”为一个可视层和一个隐层，层间存在连接，但层内的单元间不存在连接。隐层单元被训练去捕捉在可视层表现出来的高阶数据的相关性。

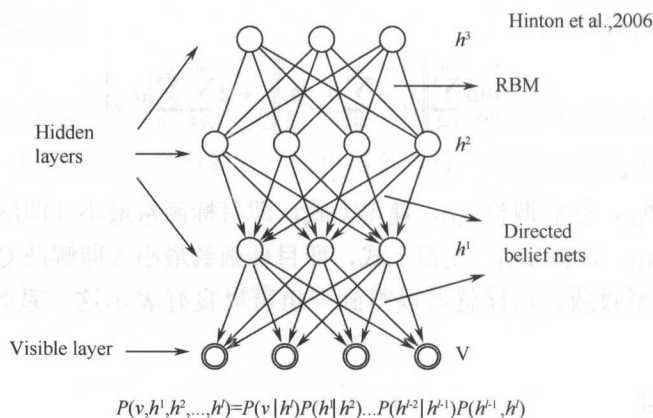


图 11.6 DBN 结构

先不考虑最顶部构成一个联想记忆（Associative Memory）的两层，一个 DBN 的连接是通过自顶向下的生成权值来指导确定的，RBM 就像一个建筑块一样，相比于传统和深度分层的 Sigmoid 信念网络，它更易于连接权值的学习。

最开始的时候，通过一个非监督贪婪逐层方法去预训练获得生成模型的权值，非监督贪婪逐层方法被 Hinton 证明是有效的，并被其称为对比分歧（Contrastive Divergence）。在这个训练阶段，在可视层会产生一个向量 v ，通过它将值传递到隐层。反过来，可视层的输入会被随机选择，以尝试去重构原始的输入信号。最后，这些新的可视的神经元激活单元将前向传递重构隐层激活单元，获得 h （在训练过程中，首先将可视向量值映射给隐单元，然后由隐单元重建可视单元，再将这些新的可视单元映射给隐单元，这样就可获取新

的隐单元。执行这种反复步骤叫做吉布斯采样)。这些后退和前进的步骤就是我们熟悉的吉布斯采样,而隐层激活单元和可视层输入之间的相关性差别就是权值更新的主要依据。

因为只需要单个步骤就可以接近最大似然学习,训练时间会显著减少。增加进网络的每一层都会改进训练数据的对数概率,我们可以理解为越来越接近能量的真实表达。这个有意义的拓展和无标签数据的使用,是任何一个深度学习应用的决定性的因素。

DBN 框架如图 11.7 所示。

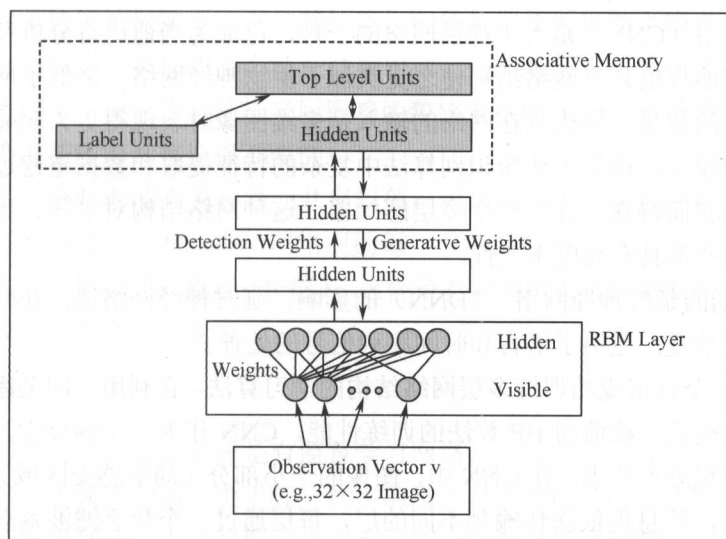


图 11.7 DBN 框架

在最高两层,权值被连接到一起,这样更低层的输出将会提供一个参考的线索或者关联给顶层,顶层就会将其联系到它的记忆内容。而在分类任务中我们最关心的是最终的判别性能。

在预训练后,DBN 可以利用带标签的数据以 BP 算法对判别性能做调整。在这里,一个标签集将被附加到顶层(推广联想记忆),通过一个自下向上学习到的识别权值获得一个网络的分类面。这个性能会比单纯的 BP 算法训练的网络好。这可以很直观地解释,DBN 的 BP 算法只需要对权值参数空间进行局部搜索,这相比于前向神经网络来说,训练较快,而且收敛的时间也较少。

DBN 的灵活性使得它的拓展比较容易。一个拓展就是卷积 DBN (Convolutional Deep Belief Network, CDBN)。DBN 并没有考虑到图像的二维结构信息,因为输入是简单地从一个图像矩阵一维向量化的。而 CDBN 就考虑到了这个问题,它利用邻域像素的空域关系,通过一个被称为卷积 RBM 的模型区达到生成模型的变换不变性,而且可以容易地变换到高维图像。

目前,和 DBN 有关的研究包括堆叠自动编码器,即用堆叠自动编码器来替换传统 DBN 里面的 RBM。这就使得可以通过同样的规则来训练产生深度多层神经网络架构,但它缺少层的参数化的严格要求。与 DBN 不同,自动编码器使用判别模型,这样这个结构就很难采

样输入采样空间，这就使得网络更难捕捉它的内部表达。但是，降噪自动编码器却能很好地避免这个问题，并且比传统的 DBN 更优秀。它通过在训练过程中添加随机的污染并堆叠产生场泛化性能。训练单一的降噪自动编码器的过程和 RBM 训练生成模型的过程一样。

11.3.2 卷积神经网络

卷积神经网络（CNN）是人工神经网络的一种，已成为当前语音分析和图像识别领域的研究热点。它的权值共享网络结构使之更类似于生物神经网络，降低了网络模型的复杂度，减少了权值的数量。该优点在网络的输入是多维图像时表现得更为明显，使图像可以直接作为网络的输入，避免了传统识别算法中复杂的特征提取和数据重建过程。卷积网络是为识别二维形状而特殊设计的一个多层感知器，这种网络结构对平移、比例缩放、倾斜或者其他形式的变形具有高度不变性。

CNN 受早期的延时神经网络（TDNN）的影响。延时神经网络通过在时间维度上共享权值降低学习复杂度，适用于语音和时间序列信号的处理。

CNN 是第一个真正成功训练多层网络结构的学习算法。它利用空间关系减少需要学习的参数数目，以提高一般前向 BP 算法的训练性能。CNN 作为一个深度学习架构提出是为了最小化数据的预处理要求。在 CNN 中，图像的一小部分（局部感受区域）作为层级结构的最低层的输入，信息再依次传输到不同的层，每层通过一个数字滤波器获得观测数据的最显著的特征。这个方法能够获取对平移、缩放和旋转不变的观测数据的显著特征，因为图像的局部感受区域允许神经元或者处理单元访问到最基础的特征，例如定向边缘或者角点。

1. 卷积神经网络的结构

卷积神经网络是一个多层的神经网络，每层由多个二维平面组成，而每个平面由多个独立神经元组成（图 11.8）。

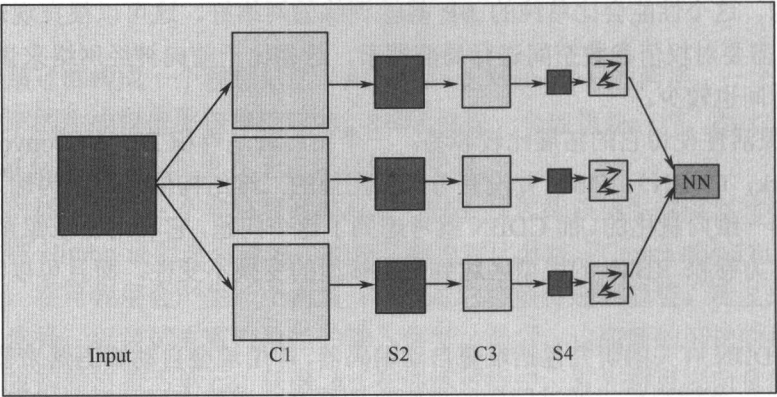


图 11.8 卷积神经网络

输入图像和三个可训练的滤波器及可加偏置进行卷积，滤波过程如图 11.8 所示，卷积后在 C1 层产生三个特征映射图，然后对特征映射图中每组的 4 个像素进行求和，加权值，加偏置，通过一个 Sigmoid 函数得到三个 S2 层的特征映射图。这些映射图再经过滤波得到 C3 层。这个层级结构再和 S2 一样产生 S4。最终，这些像素值被光栅化，并连接成一个向量输入传统的神经网络，得到输出。

一般，C 层为特征提取层，每个神经元的输入与前一层的局部感受域相连，并提取该局部特征，该局部特征被提取后，它与其他特征间的位置关系也随之确定下来；S 层是特征映射层，网络的每个计算层由多个特征映射组成，每个特征映射为一个平面，平面上所有神经元的权值相等。特征映射结构采用影响函数核大小的 Sigmoid 函数作为卷积网络的激活函数，使得特征映射具有位移不变性。

此外，由于一个映射面上的神经元共享权值，因而减少了网络自由参数的个数，降低了网络参数选择的复杂度。卷积神经网络中的每一个特征提取层（C 层）都紧跟着一个用来求局部平均与二次提取的计算层（S 层），这种特有的两次特征提取结构使网络在识别时对输入样本有较高的畸变容忍能力。

2. 训练过程

神经网络用于模式识别的主流是有监督学习网络，无监督学习网络更多的是用于聚类分析。对于有指导的模式识别，由于任一样本的类别是已知的，样本在空间的分布不再依据其自然分布倾向来划分，而是根据同类样本在空间的分布及不同类样本之间的分离程度找到一种适当的空间划分方法，或者找到一个分类边界，使得不同类样本分别位于不同的区域内。这就需要有一个长时间且复杂的学习过程，不断调整用以划分样本空间的分类边界的位置，使尽可能少的样本被划分到非同类区域中。

卷积网络在本质上是一种输入到输出的映射，它能够学习大量的输入与输出之间的映射关系，而不需要任何输入和输出之间的精确的数学表达式，只要用已知的模式对卷积网络加以训练，网络就具有输入输出对之间的映射能力。卷积网络执行的是有监督训练，所以其样本集是由形如（输入向量，理想输出向量）的向量对构成的。所有这些向量对，都应该来源于网络即将模拟的系统的实际“运行”结果。它们可以从实际运行系统中采集来的。在开始训练前，所有的权值都应该用一些不同的小随机数进行初始化。“小随机数”用来保证网络不会因权值过大而进入饱和状态，从而导致训练失败；“不同”用来保证网络可以正常地学习。实际上，如果用相同的数去初始化权值矩阵，则网络无能力学习。

训练算法与传统的 BP 算法相似，主要分为两个阶段，共 4 个步骤。

（1）向前传播阶段

① 从样本集中取一个样本 (X, Y_p) ，将 X 输入网络。

② 计算相应的实际输出 O_p 。

在此阶段，信息从输入层经过逐级变换，传送到输出层。这个过程也是网络在完成训练后正常运行时执行的过程。在此过程中，网络执行的是计算（实际上就是输入与每层的

权值矩阵相点乘，得到最后的输出结果)：

$$O_p = F_n(\dots(F_2(F_1(X_p W^{(1)}) W^{(2)}) \dots) W^{(n)})$$

(2) 向后传播阶段

① 计算实际输出 O_p 与相应的理想输出 Y_p 的差。

② 按极小化误差的方法反向传播调整权值矩阵。

3. 卷积神经网络的优点

卷积神经网络主要用来识别位移、缩放及其他形式扭曲不变性的二维图形。由于 CNN 的特征检测层通过训练数据进行学习，所以在使用 CNN 时，避免了显式的特征抽取，而隐式地从训练数据中进行学习；再者，由于同一特征映射面上的神经元权值相同，所以网络可以并行学习，这也是卷积网络相对于神经元彼此相连网络的一大优势。卷积神经网络以其局部权值共享的特殊结构在语音识别和图像处理方面有着独特的优越性，其布局更接近于实际的生物神经网络，权值共享降低了网络的复杂性，特别是多维输入向量的图像可以直接输入网络这一特点避免了特征提取和分类过程中数据重建的复杂度。

流的分类方式几乎都是基于统计特征的，这就意味着在进行分辨前必须提取某些特征。然而，显式的特征提取并不容易，在一些应用问题中也并非总是可靠的。卷积神经网络避免了显式的特征取样，隐式地从训练数据中进行学习。这使得卷积神经网络明显有别于其他基于神经网络的分类器，通过结构重组和减少权值将特征提取功能融合进多层感知器。它可以直接处理灰度图片，能够直接用于处理基于图像的分类。

卷积网络较一般神经网络在图像处理方面有如下优点：①输入图像和网络的拓扑结构能很好地吻合；②特征提取和模式分类同时进行，并同时在训练中产生；③权重共享可以减少网络的训练参数，使神经网络结构变得更简单，适应性更强。

11.4 深度学习的训练加速

深度学习模型训练需要各种技巧，例如网络结构的选取、神经元个数的设定、权重参数的初始化、学习率的调整、Mini-batch 的控制等。即便对这些技巧十分精通，实践中也需要多次训练，反复摸索尝试。此外，深层模型参数多，计算量大，训练数据的规模更大，需要消耗大量计算资源。如果可以让训练加速，就可以在同样的时间内多尝试几个新方案，多调试几组参数，工作效率会明显提升，对于大规模的训练数据和模型来说，更可以将难以完成的任务变成可能。本节主要探讨深层模型的训练加速方法。

11.4.1 GPU 加速

矢量化编程是提高算法速度的一种有效方法。为了提升特定数值运算操作（如矩阵相

乘、矩阵相加、矩阵—向量乘法等)的速度,数值计算和并行计算的研究人员已经努力了几十年。矢量化编程强调单一指令并行操作多条相似数据,形成单指令流多数据流(SIMD)的编程泛型。深层模型的算法,如 BP、Auto-Encoder、CNN 等,都可以写成矢量化的形式。然而,在单个 CPU 上执行时,矢量运算会被展开成循环的形式,本质上还是串行执行。

GPU (Graphic Process Units, 图形处理器)的众核体系结构包含几千个流处理器,可将矢量运算并行化执行,大幅缩短计算时间。随着 nVIDIA、AMD 等公司不断推进其 GPU 的大规模并行架构支持,面向通用计算的 GPU (General-Purposed GPU, GPGPU)已成为加速可并行应用程序的重要手段。得益于 GPU 众核 (Many-core) 体系结构,程序在 GPU 系统上的运行速度相较于单核 CPU 往往提升几十倍乃至上千倍。目前 GPU 已经发展到了较为成熟的阶段,受益最大的是科学计算领域,典型的成功案例包括多体问题 (N-Body Problem)、蛋白质分子建模、医学成像分析、金融计算、密码计算等。

利用 GPU 来训练深度神经网络,可以充分发挥其数以千计计算核心的高效并行计算能力,在使用海量训练数据的场景下,所耗费的时间大幅缩短,占用的服务器也更少。如果针对适当的深度神经网络进行合理优化,一块 GPU 卡的计算能力可相当于数十甚至上百台 CPU 服务器的计算能力,因此 GPU 已经成为业界在深度学习模型训练方面的首选解决方案。

11.4.2 数据并行

数据并行是指对训练数据做切分,同时采用多个模型实例,对多个分片的数据并行训练(图 11.9)。

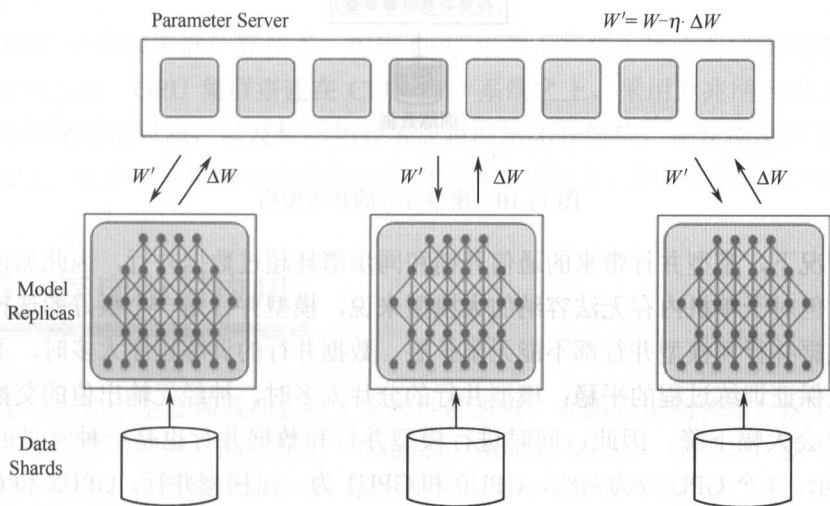


图 11.9 数据并行的基本架构

要完成数据并行需要做参数交换,通常由一个参数服务器 (Parameter Server) 来帮助

完成。在训练的过程中，多个训练过程相互独立，训练的结果，即模型的变化量 ΔW 需要汇报给参数服务器，由参数服务器负责更新为最新的模型 $W' = W - \eta \cdot \Delta W$ ，然后将最新的模型 W' 分发给训练程序，以便从新的起点开始训练。

数据并行有同步模式和异步模式之分。同步模式中，所有训练程序同时训练一个批次的训练数据，完成后经过同步，再同时交换参数。参数交换完成后所有的训练程序就有了共同的新模型作为起点，再训练下一个批次。而异步模式中，训练程序完成一个批次的训练数据，立即和参数服务器交换参数，不考虑其他训练程序的状态。异步模式中一个训练程序的最新结果不会立刻体现在其他训练程序中，直到它们进行下次参数交换。

参数服务器只是一个逻辑上的概念，不一定部署为独立的一台服务器。有时候它会附属在某一个训练程序上；有时也会将参数服务器按照模型划分为不同的分片，分别部署。

11.4.3 模型并行

模型并行是指将模型拆分成几个分片，由几个训练单元分别持有，共同协作完成训练（图 11.10）。当一个神经元的输入来自另一个训练单元上的神经元的输出时，会产生通信开销。

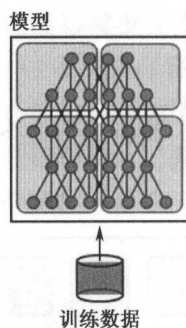


图 11.10 模型并行的基本架构

多数情况下，模型并行带来的通信开销和同步消耗超过数据并行，因此加速比也不及数据并行。但对于单机内存无法容纳的大模型来说，模型并行是一个很好的选择。令人遗憾的是，数据并行和模型并行都不能无限扩展。数据并行的训练程序太多时，不得不减小学习率，以保证训练过程的平稳；模型并行的分片太多时，神经元输出值的交换量会急剧增加，效率会大幅下降。因此，同时进行模型并行和数据并行也是一种常见的方案。如图 11.11 所示，4 个 GPU 分为两组，GPU0 和 GPU1 为一组模型并行，GPU2 和 GPU3 为另一组，每组模型并行在计算过程中交换输出值和残差。两组 GPU 之间形成数据并行，Mini-batch 结束后交换模型权重，考虑到模型黑色部分由 GPU0 和 GPU2 持有，而白色部分由 GPU1 和 GPU3 持有，因此只有同色的 GPU 之间需要交换权重。

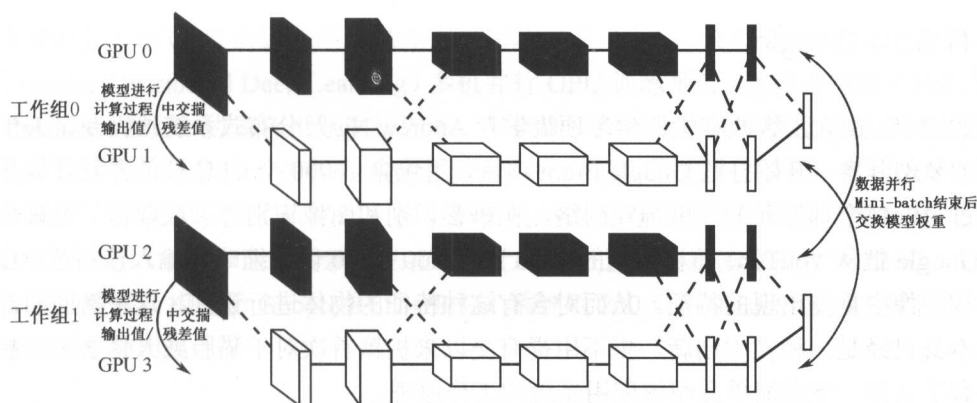


图 11.11 4 个 GPU 的数据并行和模型并行混合架构

11.4.4 计算集群

搭建 CPU 集群用于深度神经网络模型训练也是业界常用的解决方案，其优势在于利用大规模分布式计算集群的强大计算能力，利用模型可分布式存储、参数可异步通信的特点，达到快速训练深层模型的目的。

CPU 集群方案的基本架构包含用于执行训练任务的 Worker、用于分布式存储分发模型参数服务器（Parameter Server）和用于协调整体任务的主控程序（Master）。CPU 集群方案适合训练 GPU 内存难以容纳的大模型，以及稀疏连接神经网络。Andrew Ng 和 Jeff Dean 在 Google 用 1000 台 CPU 服务器，完成了模型并行和 Downpour SGD 数据并行的深度神经网络训练。

结合 GPU 计算和集群计算技术，构建 GPU 集群正在成为加速大规模深度神经网络训练的有效解决方案。GPU 集群搭建在 CPU-GPU 系统之上，采用万兆网卡或 Infiniband 等更加快速的网络通信设施，以及树形拓扑等逻辑网络拓扑结构。在发挥出单节点较高计算能力的基础上，充分挖掘集群中多台服务器的协同计算能力，进一步加速大规模训练任务。

11.5 深度学习应用

深度学习在几个主要领域都获得了突破性的进展：在语音识别领域，深度学习用深层模型替换声学模型中的混合高斯模型（Gaussian Mixture Model, GMM），获得了 30% 左右的相对错误率降低；在图像识别领域，通过构造深度卷积神经网络，将 Top5 错误率由 26% 大幅降低至 15%，又通过加大加深网络结构，进一步降低到 11%；在自然语言处理领域，深度学习基本获得了与其他方法水平相当的结果，但可以免去烦琐的特征提取步骤。可以说到目前为止，深度学习是最接近人类大脑的智能学习方法。下面将介绍深度学习在业界的应用。

11.5.1 Google

2012年,由人工智能和机器学习顶级学者 Andrew Ng 及分布式系统顶级专家 Jeff Dean 领衔的梦幻阵容,开始打造 Google Brain 项目,用包含 16000 个 CPU 核的并行计算平台训练超过 10 亿个神经元的深度神经网络,在语音识别和图像识别等领域取得了突破性的进展。Google 把从 YouTube 随机挑选的 1000 万张 200×200 像素缩略图输入该系统,让计算机寻找图像中重复出现的特征,从而对含有这种特征的物体进行识别。这种新的面部识别方式本身已经是一种技术创新,更不用提有史以来机器首次对于猫脸或人体这种“高级概念”有了认知。下面简单介绍该应用系统的工作原理。

在开始分析数据之前,工作人员不会教授系统或者向系统输入任何诸如“脸、肢体、猫的长相是什么样子”这类信息。一旦系统发现了重复出现的图像信息,计算机就创建出“图像地图”,该地图稍后会帮助系统自动检测与前述图像信息类似的物体。Google 把它命名为“神经系统”,旨在向神经生物学中的一个经典理论致敬。这个理论指出,人类大脑颞叶皮层的某些神经元是专门用来识别面部、手等这类对象的。

以往传统的面部识别技术,一般都是由研究者先在计算机中通过定义识别对象的形状边缘等信息来“教会”计算机该对象的外观应该如何,然后计算机对包含同类信息的图片做出标识,从而达到“识别”的结果。Jeff Dean 博士(“神经系统”参与者)表示,在 Google 的这个新系统里,工作人员从不向计算机描述“猫长什么样”这类信息,计算机基本上靠自己产生出“猫”这一概念。

截至目前,这个系统还不完美。但它取得的成功有目共睹,Google 已经将该项目从 Google X 中独立出来,现在由总公司的搜索及商业服务小组继续引领完成。Google 的目标是宏伟的,它希望能开创一种全新的算法,并将其应用于图像识别、语言识别,以及机器语言翻译等更广阔的领域。

11.5.2 百度

在深度学习方面,百度已经在学术理论、工程实现、产品应用等多个领域取得了显著的进展,已经成为业界推动“大数据驱动的人工智能”的领导者之一。

在图像技术应用中,传统的从图像到语义的转换是极具挑战性的课题,业界称其为语义鸿沟。百度深度学习算法构造出一个多层非线性层叠式神经网络,能够很好地模拟视觉信号从视网膜开始逐层处理传递,直至大脑深处的整个过程。这样的学习模式能够以更高的精度和更快的速度跨越语义鸿沟,让机器快速地对图像中可能蕴含的成千上万种语义概念进行有效识别,进而确定图片的主题。在人脸识别方面,最困难的是识别照片中的人是谁或者通过照片寻找相似的人。百度在深度学习的基础上,借鉴认知学中的一些概念与方法,探索出了独特的相似度量学习方法来寻找图像的相似性和关联,能够做到举一反三。

在深度神经网络训练方面,伴随着计算广告、文本、图像、语音等训练数据的快速增

长,传统的基于单 GPU 的训练平台已经无法满足需求。为此,百度搭建了 Paddle (Parallel Asynchronous Distributed Deep Learning) 多机并行 GPU 训练平台。数据分布到不同的机器,通过 Parameter Server 协调各机器进行训练,多机训练使得大数据的模型训练成为可能。

在算法方面,单机多卡并行训练算法研发,难点在于通过并行提高计算速度一般会降低收敛速度。百度则研发了新算法,在不影响收敛速度的条件下计算速度图像提升至 2.4 倍,语音提升至 1.4 倍,这使得新算法在单机上的收敛速度达到图像提升至 12 倍、语音提升至 7 倍的效果。相比于 Google 的 DistBelief 系统用 200 台机器加速约 7.3 倍而言,百度的算法优势更加明显。

11.5.3 腾讯 Mariana

面对机遇和挑战,腾讯打造了深度学习平台 Mariana,该平台包括三个框架:深度神经网络的 GPU 数据并行框架,深度卷积神经网络的 GPU 数据并行和模型并行框架,以及 DNN CPU 集群框架。基于上述三个框架,Mariana 具有多种特性:

- ① 支持并行加速,针对多种应用场景,解决深度学习训练极慢的问题;
- ② 通过模型并行,支持大模型;
- ③ 提供默认算法的并行实现以减少新算法开发量,简化实验过程;
- ④ 面向语音识别、图像识别、广告推荐等众多应用领域。

腾讯深度学习平台 Mariana 重点研究多 GPU 卡的并行化技术,完成 DNN 的数据并行框架,以及 CNN 的模型并行和数据并行框架。数据并行指将训练数据划分为多份,每份数据有一个模型实例进行训练,再将多个模型实例产生的梯度合并后更新模型。模型并行指将模型划分为多个分片,每个分片在一台服务器上,全部分片协同对一份训练数据进行训练。

DNN 的数据并行框架已经成功应用在微信语音识别中。微信中语音识别功能的入口是语音输入法、语音开放平台以及长按语音消息转文本等。对微信语音识别任务,通过 Mariana,识别准确率获得了极大的提升,目前识别能力已经达到业界一流水平。同时可以满足语音业务海量的训练样本需求,通过缩短模型更新周期,使微信语音业务可以及时满足各种新业务需求。

同时,Mariana 的 CNN 模型并行和数据并行框架,针对 ImageNet 图像分类问题,在单机 4 GPU 卡配置下,获得了相对于单卡 2.52 倍的加速,并支持更大模型,在 ImageNet 2012 数据集中获得了 87% 的 Top5 准确率。此外,Mariana 在广告推荐及个性化推荐等领域,也正在积极探索和实验中。

11.6 练习题

1. 简述深度学习、数据挖掘和机器学习之间的关系。
2. 深度学习的常用方法和模型有哪些？
3. 深度学习的训练加速方式有哪几种？
4. 简述业界深度学习的应用。

参考文献

- [1] Geoffery E. Hinton, Salakhutdinov RR. Reducing the dimensionality of data with neural networks. Science, 2006 Jul 28, 313(5786):504-7.
- [2] Alex Krizhevsky, Ilya Sutskever, Geoffrey E Hinton. ImageNet Classification with Deep Convolutional Neural Networks. NIPS, 2012.
- [3] Q.V. Le, M.A. Ranzato, R. Monga, M. Devin, K. Chen, G.S. Corrado, J. Dean, A.Y. Ng. Building high-level features using large scale unsupervised learning. ICML, 2012.
- [4] Stanford deep learning tutorial. http://deeplearning.stanford.edu/wiki/index.php/UFLDL_Tutorial
- [5] A Primer on Deep Learning. <http://www.datarobot.com/blog/a-primer-on-deep-learning/>
- [6] Bruno A. Olshausen & David J. Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. Nature, Vol 381, 13 June, 1996. http://www.cs.ubc.ca/~little/cpsc425/olshausen_field_nature_1996.pdf
- [7] Back propagation algorithm. http://ufldl.stanford.edu/wiki/index.php/Backpropagation_Algorithm
- [8] Support Vector Machine. http://en.wikipedia.org/wiki/Support_vector_machine
- [9] Logistic Regression. http://en.wikipedia.org/wiki/Logistic_regression
- [10] Deep Networks Overview. http://ufldl.stanford.edu/wiki/index.php/Deep_Networks:_Overview
- [11] Y. LeCun and Y. Bengio. Convolutional networks for images, speech, and time-series. In M. A. Arbib, editor, The Handbook of Brain Theory and Neural Networks. MIT Press, 1995.
- [12] Introduction to Convolutional neural network. http://en.wikipedia.org/wiki/Convolutional_neural_network
- [13] Dean J., Corrado G.S., Monga R., et al. Large Scale Distributed Deep Networks. In Proceedings of the Neural Information Processing Systems (NIPS'12) (Lake Tahoe, Nevada, United States, December 3–6, 2012). Curran Associates, Inc, 57 Morehouse Lane, Red Hook, NY, 2013, 1223-1232.
- [14] Povey, D., Ghoshal, A. Boulianne, G., et al. The Kaldi Speech Recognition Toolkit. in

- Proceedings of IEEE 2011 Workshop on Automatic Speech Recognition and Understanding(ASRU 2011) (Hilton Waikoloa Village, Big Island, Hawaii, US, December 11-15, 2011). IEEE Signal Processing Society. IEEE Catalog No.: CFP11SRW-USB.
- [15] Krizhevsky, A., Sutskever, I., and Hinton, G.E. ImageNet Classification with Deep Convolutional Neural Networks. In Proceedings of the Neural Information Processing Systems (NIPS'12) (Lake Tahoe, Nevada, United States, December 3–6, 2012). Curran Associates, Inc, 57 Morehouse Lane, Red Hook, NY, 2013, 1097-1106.
 - [16] Krizhevsky, A. Parallelizing Convolutional Neural Networks. in tutorial of IEEE Conference on Computer Vision and Pattern Recognition (CVPR 2014). Columbus, Ohio, USA, June 23-28, 2014.
 - [17] Jia, Y. Q. Caffe: An Open Source Convolutional Architecture for Fast Feature Embedding. 2013.<http://caffe.berkeleyvision.org>
 - [18] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley and Y. Bengio. Theano: A CPU and GPU Math Expression Compiler. Proceedings of the Python for Scientific Computing Conference (SciPy) 2010. June 30 - July 3, Austin, TX.
 - [19] Coates, A., Huval, B., Wang, T., Wu, D. J., Ng, A. Y. Deep learning with COTS HPC systems. In Proceedings of the 30th International Conference on Machine Learning (ICML'13) (Atlanta, Georgia, USA, June 16–21, 2013). JMLR: W&CP, 2013, 28(3): 1337-1345.
 - [20] Yadan, O., Adams, K., Taigman, Y., Ranzato, M. A. Multi-GPU Training of ConvNets. arXiv:1312.5853v4, 2014.
 - [21] Kaiyu. Large-scale Deep Learning at Baidu. ACM International Conference on Information and Knowledge Management (CIKM 2013).
 - [22] Geoffrey E. Hinton, Simon Osindero, Yee-Whye The. A fast learning algorithm for deep belief nets. Neural Compute, 2006,18(7): 1527-54 .
 - [23] Andrew Ng. Machine Learning and AI via Brain simulations, <https://forum.stanford.edu/events/2011slides/plenary/2011plenaryNg.pdf>
 - [24] Geoffrey Hinton.UCLTutorial on: Deep Belief Nets.
 - [25] Krizhevsky, Alex. ImageNet Classification with Deep Convolutional Neural Networks. Retrieved 17 November 2013.
 - [26] Bengio. Learning Deep Architectures for AI. http://www.iro.umontreal.ca/~bengioy/papers/ftml_book.pdf.
 - [27] Introduction to Deep Learning. http://en.wikipedia.org/wiki/Deep_learning
 - [28] Andrew Ng's talk video. <http://techtalks.tv/talks/machine-learning-and-ai-via-brain-simulations/57862/>.

第12章

电子商务与社会化网络大数据分析

本章内容比较多，而且和当前互联网最赚钱的行业关系密切。无论是电子商务、计算广告还是社会化网络都是很复杂的。针对每个领域，本章主要介绍如下内容：

①介绍在该领域做得比较好的互联网公司，以及这些公司的哪些产品中使用了与大数据相关的技术。②介绍相关的算法。③介绍应用实例。

针对电子商务这一部分，将介绍常用的推荐算法并模拟一些测试数据。关于计算广告部分，将介绍目前的主流 RTB (Real Time Bidding)，RTB 即实时竞价，是目前主流的广告交易模式，国内外做得比较好的有百度、阿里妈妈、Google 等；此外还将介绍需求方平台 (Demand-Side Platform, DSP)，通过模拟 RTB 程序让读者知道实时竞价相关的流程，并重点讲解广告排序相关的技术点。最后关于社交网络，将主要介绍关系挖掘，比如二度人脉的挖掘、语言流行度等。

12.1 推荐系统简介

推荐系统在人们的生活中无处不在，比如顾客买包子的时候，老板经常会问顾客要不要来杯豆浆，这就是一种简单的推荐。互联网的发展大大扩展了推荐系统的应用，如亚马逊的商品推荐、Facebook 的好友推荐、Digg 的文章推荐、豆瓣的豆瓣猜，Last.fm 和豆瓣 FM 的音乐推荐、Gmail 里的广告等。在如今互联网信息过载的情况下，信息消费者想方便地找到自己感兴趣的内容，信息生产者则想将自己的内容推送给最合适的目标用户。而推

荐系统正是这两者的中介。那怎么衡量一个推荐系统的好坏呢？有哪些具体的指标呢？常用的算法主要有哪些呢？电子商务中具体的推荐系统的应用有哪些呢？

12.1.1 推荐系统的评判标准

首先要明确什么是好的推荐系统。可以通过如下几个指标来判定。

① 用户满意度：描述用户对推荐结果的满意程度，这是推荐系统最重要的指标。一般通过对用户进行问卷调查或者监测用户线上行为数据获得。

② 预测准确度：描述推荐系统预测用户行为的能力。一般通过离线数据集上算法给出的推荐列表和用户行为的重合率来计算。重合率越大，则准确率越高。

③ 覆盖率：描述推荐系统对物品长尾的发掘能力。一般通过所有推荐物品占总物品的比例和所有物品被推荐的概率分布来计算。比例越大，概率分布越均匀，则覆盖率越高。

④ 多样性：描述推荐系统中推荐结果能否覆盖用户不同的兴趣领域。一般通过推荐列表中物品两两之间不相似性来计算。物品之间越不相似，则多样性越好。

⑤ 新颖性：如果用户没有听说过推荐列表中的大部分物品，则说明该推荐系统的新颖性较好。可以通过推荐结果的平均流行度和对用户进行问卷调查来获得。

⑥ 惊喜度：如果推荐结果和用户的历史兴趣不相似，但让用户很满意，则可以说这是一个让用户惊喜的推荐。可以定性地通过推荐结果与用户历史兴趣的相似度和用户满意度来衡量。

简而言之，一个好的推荐系统就是在推荐准确的基础上，给所有用户推荐的物品尽量广泛（挖掘长尾），给单个用户推荐的物品尽量覆盖多个类别，同时不要给用户推荐太多热门物品，最重要的则是能让用户看到推荐后有种相见恨晚的感觉。

12.1.2 推荐系统的分类

推荐系统是建立在大量有效数据之上的，背后的算法思想有很多种，可以从处理的数据入手进行分类。

互联网上的用户行为千千万万，从简单的网页浏览到复杂的评价、下单，其中蕴含了大量的用户反馈信息，通过对这些行为的分析，便能推知用户的兴趣爱好。而这其中最基础的就是“协同过滤算法”。

协同过滤算法分为两种：基于用户（UserCF）和基于物品（ItemCF）。所谓基于用户，就是根据用户对物品的行为，找出兴趣爱好相似的一些用户，将其中一个用户喜欢的东西推荐给另一个用户。基于物品就是先找出相似的物品。怎么找呢？也是看用户的喜好，如果同时喜欢某两个物品的人比较多，就可以认为这两个物品相似。最后只要给用户推荐和他原有喜好类似的物品即可。

至于什么时候用 UserCF，什么时候用 ItemCF，这要视情况而定。一般来说，UserCF 更接近于社会化推荐，适用于用户少、物品多、时效性较强的场合，比如 Digg 的文章推荐；而 ItemCF 则更接近于个性化推荐，适用于用户多、物品少的场合，比如豆瓣的豆瓣猜、豆瓣 FM。

协同过滤算法也有不少缺点，最明显的一个就是热门物品的干扰。举个例子，协同过滤算法经常会导致两个不同领域的最热门物品之间具有较高的相似度，这样很可能会给喜欢《算法导论》的读者推荐《哈利波特》。显然，这不科学。要避免这种情况就得从物品的内容数据入手，内容过滤算法就是其中一种方法。

除了协同过滤算法之外，隐语义模型 (LFM) 应用得也比较多，它基于用户行为对物品进行自动聚类，从而将物品按照多个维度、多个粒度分门别类，然后根据用户喜欢的物品类别进行推荐。这种基于机器学习的方法在很多指标上优于协同过滤算法，但性能不太完善，一般可以先通过其他算法得出推荐列表，再由 LFM 进行优化。

12.1.3 在线推荐系统常用算法介绍

推荐算法是整个推荐系统中最核心、最关键的部分，在很大程度上决定了推荐系统性能的优劣。目前，主要的推荐算法包括：基于内容推荐、协同过滤推荐、基于关联规则推荐、基于效用推荐、基于知识推荐和组合推荐。

1. 基于内容推荐

基于内容推荐 (Content-based Recommendation) 是信息过滤技术的延续与发展，它是基于项目的内容信息做出推荐的，而不需要依据用户对项目的评价意见，更多地需要用机器学习的方法从关于内容的特征描述的事例中得到用户的兴趣资料。在基于内容的推荐系统中，项目或对象是通过相关特征的属性来定义的，系统基于用户评价对象的特征，学习用户的兴趣，考察用户资料与待预测项目的相匹配程度。用户的资料模型取决于所用学习方法，常用的有决策树、神经网络和基于向量的表示方法等。基于内容的用户资料需要用户的历史数据，用户资料模型可能随着用户偏好的改变而发生变化。

基于内容推荐算法的优点有：

- ① 不需要其他用户的数据，没有冷开始问题和稀疏问题。
- ② 能为具有特殊兴趣爱好的用户进行推荐。
- ③ 能推荐新的或不是很流行的项目，没有新项目问题。
- ④ 通过列出推荐项目的内容特征，可以解释为什么推荐那些项目。
- ⑤ 已有比较好的技术，如关于分类学习方面的技术已相当成熟。

其缺点是要求内容容易抽取成有意义的特征，特征内容有良好的结构性，并且用户的口味必须能够用内容特征形式来表达，不能显式地得到其他用户的判断情况。

2. 协同过滤推荐

协同过滤推荐 (Collaborative Filtering Recommendation) 技术是推荐系统中应用最早和最为成功的技术之一。它一般采用最近邻技术, 利用用户的历史喜好信息计算用户之间的距离, 然后利用目标用户的最近邻用户对商品的加权评价来预测目标用户对特定商品的喜好程度, 从而根据这一喜好程度来对目标用户进行推荐。协同过滤最大的优点是对推荐对象没有特殊的要求, 能处理非结构化的复杂对象, 如音乐、电影。

协同过滤基于这样的假设: 为一用户找到他真正感兴趣的内容的好方法是首先找到与此用户有相似兴趣的其他用户, 然后将他们感兴趣的内容推荐给此用户。其基本思想非常易于理解, 在日常生活中, 人们往往会根据好朋友的推荐来进行一些选择。协同过滤正是把这一思想运用到电子商务推荐系统中, 基于其他用户对某一内容的评价来向目标用户进行推荐。

基于协同过滤的推荐系统可以说是从用户的角度来进行相应推荐的, 而且是自动的, 即用户获得的推荐是系统从购买模式或浏览行为等隐式获得的, 不需要用户努力地找到符合自己兴趣的推荐信息, 如填写一些调查表格等。

和基于内容的过滤方法相比, 协同过滤具有如下优点:

- ① 能够过滤难以进行机器自动内容分析的信息, 如艺术品、音乐等。
- ② 共享其他人的经验, 避免了内容分析的不完全和不精确, 并且能够基于一些复杂的、难以表述的概念 (如信息质量、个人品味) 进行过滤。
- ③ 有推荐新信息的能力。可以发现内容上完全不相似的信息, 用户对推荐信息的内容事先是预料不到的。这也是协同过滤和基于内容的过滤一个较大的差别, 基于内容的过滤推荐的很多都是用户本来就熟悉的内容, 而协同过滤可以发现用户潜在的但自己尚未发现的兴趣偏好。
- ④ 能够有效地使用其他相似用户的反馈信息, 减少用户的反馈量, 加快个性化学习的速度。

虽然协同过滤作为一种典型的推荐技术有一定的应用, 但协同过滤仍有许多问题需要解决。最典型的问题有稀疏问题 (Sparsity) 和可扩展问题 (Scalability)。

3. 基于关联规则推荐

基于关联规则推荐 (Association Rule-based Recommendation) 是以关联规则为基础, 把已购商品作为规则头, 规则体为推荐对象。关联规则挖掘可以发现不同商品在销售过程中的相关性, 在零售业中已经得到了成功的应用。管理规则就是在一个交易数据库中统计购买了商品集 X 的交易中有多大比例的交易同时购买了商品集 Y , 其直观的意义就是用户在购买某些商品的时候有多大倾向去购买另外一些商品。比如, 很多人会在购买牛奶的同时购买面包。

算法的第一步——关联规则地发现最为关键且最耗时, 是算法的瓶颈, 但可以离线进

行。其次，商品名称的同义性问题也是关联规则的一个难点。

4. 基于效用推荐

基于效用推荐 (Utility-based Recommendation) 建立在用户使用项目的效用情况的基础上，其核心问题是怎么样为每一个用户创建一个效用函数。因此，用户资料模型在很大程度上是由系统所采用的效用函数决定的。基于效用推荐的好处是它能把非产品的属性，如提供商的可靠性 (Vendor Reliability) 和产品的可得性 (Product Availability) 等考虑到效用计算中。

5. 基于知识推荐

基于知识推荐 (Knowledge-based Recommendation) 在某种程度上可以看成一种推理 (Inference) 技术，它不是基于用户需要和偏好进行推荐的。基于知识的方法因所用的功能知识不同而有明显区别。效用知识 (Functional Knowledge) 是一种关于一个项目如何满足某一特定用户的知识，因此能解释需要和推荐的关系，所以用户资料可以是任何能支持推理的知识结构，它可以是用户已经规范化的查询，也可以是一个更详细的用户需要的表示。

6. 组合推荐

由于各种推荐算法都有优缺点，所以在实际中，组合推荐 (Hybrid Recommendation) 经常被采用。研究和应用最多的是内容推荐和协同过滤推荐的组合。最简单的做法就是分别用基于内容的方法和协同过滤推荐方法产生一个推荐预测结果，然后用某种方法组合其结果。尽管在理论上有很多种推荐组合方法，但在某一具体问题中并不见得都有效，组合推荐一个最重要的原则就是通过组合能避免或弥补各种推荐技术的缺点。

在组合方式上，有研究人员提出了 7 种组合思路。

① 加权 (Weight): 加权多种推荐技术结果。

② 变换 (Switch): 根据问题背景和实际情况或要求决定变换采用不同的推荐技术。

③ 混合 (Mixed): 同时采用多种推荐技术给出多种推荐结果为用户提供参考。

④ 特征组合 (Feature Combination): 组合来自不同推荐数据源的特征供另一种推荐算法使用。

⑤ 层叠 (Cascade): 先用一种推荐技术产生一种粗糙的推荐结果，再利用第二种推荐技术在此推荐结果的基础上做出更精确的推荐。

⑥ 特征扩充 (Feature Augmentation): 将一种技术产生附加的特征信息嵌入另一种推荐技术的特征输入中。

⑦ 元级别 (Meta-level): 用一种推荐方法产生的模型作为另一种推荐方法的输入。

几种主要推荐算法的对比见表 12.1。

表 12.1 推荐算法对比

推荐算法	优点	缺点
基于内容推荐	① 推荐结果直观，容易解释 ② 不需要领域知识	① 稀疏问题 ② 新用户问题 ③ 复杂属性不好处理 ④ 要有足够数据构造分类器
协同过滤推荐	① 新异兴趣发现，不需要领域知识 ② 随着时间推移性能提高 ③ 推荐个性化、自动化程度高 ④ 能处理复杂的非结构化对象	① 稀疏问题 ② 可扩展性问题 ③ 新用户问题 ④ 质量取决于历史数据集 ⑤ 系统开始时推荐质量差
基于关联规则推荐	① 能发现新兴趣点 ② 不需要领域知识	① 规则抽取难、耗时 ② 产品名同义性问题 ③ 个性化程度低
基于效用推荐	① 无冷开始和稀疏问题 ② 对用户偏好变化敏感 ③ 能考虑非产品特性	① 用户必须输入效用函数 ② 推荐是静态的，灵活性差 ③ 属性重叠问题
基于知识推荐	① 能把用户需求映射到产品上 ② 能考虑非产品属性	① 知识难获得 ② 推荐是静态的

12.1.4 相关算法知识

本节主要介绍距离和相似度量，以及数据的归一化。

1. 距离和相似度量

在数据分析和数据挖掘的过程中，经常需要知道个体间差异的大小，进而评价个体的相似性和类别。最常见的是数据分析中的相关分析、数据挖掘中的分类和聚类算法，如 K 最近邻 (KNN) 和 K 均值 (K -Means)。

为了方便下面的解释和举例，先设定要比较 X 个体和 Y 个体间的差异，它们都包含了 N 维特征，即 $X=(x_1,x_2,x_3,\cdots,x_n)$ ， $Y=(y_1,y_2,y_3,\cdots,y_n)$ 。下面来看看可以采用哪些方法来衡量两者的差异，主要分为距离度量和相似度量。

(1) 距离度量

距离 (Distance) 度量用于衡量个体在空间上存在的距离，距离越远说明个体间的差异越大。

① 欧几里得距离 (Euclidean Distance)。欧几里得距离又称欧氏距离，是最常见的距离度量，衡量的是多维空间中各个点之间的绝对距离。公式如下：

$$\text{dist}(X,Y)=\sqrt{\sum_{i=1}^n(x_i-y_i)^2}$$

因为计算基于各维度特征的绝对数值，所以欧氏距离需要保证各维度指标在相同的刻度级别，比如对身高（cm）和体重（kg）两个单位不同的指标使用欧式距离可能使结果失效。

② 明可夫斯基距离（Minkowski Distance）。明可夫斯基距离是欧氏距离的推广，是对多个距离度量公式概括性的表述。公式如下：

$$\text{dist}(X, Y) = \left(\sum_{i=1}^n |x_i - y_i|^p \right)^{1/p}$$

这里的 p 是一个变量，当 $p=2$ 时就得到了欧氏距离。

③ 曼哈顿距离（Manhattan Distance）。曼哈顿距离来源于城市区块距离，是将多个维度上的距离进行求和后的结果。上面的明可夫斯基距离公式中 $p=1$ 时即为曼哈顿距离公式，具体如下：

$$\text{dist}(X, Y) = \sum_{i=1}^n |x_i - y_i|$$

④ 切比雪夫距离（Chebyshev Distance）。切比雪夫距离起源于国际象棋中国王的走法。扩展到多维空间，其实切比雪夫距离就是当 p 趋向于无穷大时的明可夫斯基距离，公式如下：

$$\text{dist}(X, Y) = \lim_{p \rightarrow \infty} \left(\sum_{i=1}^n |x_i - y_i|^p \right)^{1/p} = \max |x_i - y_i|$$

曼哈顿距离、欧氏距离和切比雪夫距离都是明可夫斯基距离在特殊条件下的应用。

⑤ 马哈拉诺比斯距离（Mahalanobis Distance）。既然欧几里得距离无法忽略指标度量的差异，所以在使用欧氏距离之前需要对底层指标进行数据的标准化，而基于各指标维度进行标准化后再使用欧氏距离就衍生出来另外一个距离度量——马哈拉诺比斯距离（Mahalanobis Distance），简称马氏距离。

（2）相似度度量

相似度（Similarity）度量，即计算个体间的相似程度。与距离度量相反，相似度度量的值越小，说明个体间相似度越小，差异越大。

① 向量空间余弦相似度（Cosine Similarity）。余弦相似度用向量空间中两个向量夹角的余弦值衡量两个个体间的差异。相比于距离度量，余弦相似度更加注重两个向量在方向上的差异，而非距离或长度上。公式如下：

$$\text{sim}(X, Y) = \cos \theta = \frac{\vec{x} \cdot \vec{y}}{\|\vec{x}\| \cdot \|\vec{y}\|}$$

② 皮尔森相关系数（Pearson Correlation Coefficient）。皮尔森相关系数即相关分析中的相关系数 r ，分别对 X 和 Y 基于自身总体标准化后计算空间向量的余弦夹角。公式如下：

$$r(X, Y) = \frac{n \sum xy - \sum x \sum y}{\sqrt{n \sum x^2 - (\sum x)^2} \cdot \sqrt{n \sum y^2 - (\sum y)^2}}$$

③ Jaccard 相似系数 (Jaccard Coefficient)。Jaccard 相似系数主要用于计算符号度量或布尔值度量的个体间的相似度, 因为个体的特征属性都是由符号度量或者布尔值标识的, 因此无法衡量差异具体值的大小, 只能获得“是否相同”这个结果, 所以 Jaccard 相似系数只关心个体间共同具有的特征是否一致这个问题。如果比较 X 与 Y 的 Jaccard 相似系数, 只比较 x_n 和 y_n 中相同的个数, 公式如下:

$$\text{Jaccard}(X, Y) = \frac{X \cap Y}{X \cup Y}$$

④ 调整余弦相似度 (Adjusted Cosine Similarity)。虽然余弦相似度对个体间存在的偏见可以进行一定的修正, 但是因为其只能分辨个体在维之间的差异, 没法衡量每个维数值的差异, 所以会导致这样一种情况: 比如用户对内容评分, 采用 5 分制, X 和 Y 两个用户对两个内容的评分分别为 (1,2) 和 (4,5), 使用余弦相似度得出的结果是 0.98, 两者极为相似, 但从评分上看 X 似乎不喜欢这两个内容, 而 Y 比较喜欢。余弦相似度对数值的不敏感导致了结果的误差, 需要修正这种不合理性, 因此出现了调整余弦相似度, 即所有维度上的数值都减去一个均值, 比如 X 和 Y 的评分均值都是 3, 那么调整后为 (-2,-1) 和 (1,2), 再用余弦相似度计算, 得到 -0.8, 相似度为负值并且差异不小, 但显然更加符合现实。

(3) 欧氏距离与余弦相似度

欧氏距离是最常见的距离度量, 而余弦相似度则是最常见的相似度度量, 很多距离度量和相似度度量都是这两者的变形和衍生, 所以下面重点比较两者在衡量个体差异时实现方式和应用环境上的区别。

借助三维坐标系来看一下欧氏距离和余弦相似度的区别, 如图 12.1 所示。

从图 12.2 中可以看出, 欧氏距离衡量的是空间各点间的绝对距离, 跟各个点所在的位置坐标 (即个体特征维度的数值) 直接相关; 而余弦相似度衡量的是空间向量的夹角, 体现的是方向上的差异, 而不是位置。如果保持 A 点的位置不变, B 点朝原方向远离坐标轴原点, 那么这个时候余弦相似度 $\cos\theta$ 是保持不变的, 因为夹角不变, 而 A 、 B 两点的距离显然在发生改变, 这就是欧氏距离和余弦相似度的不同之处。

根据欧氏距离和余弦相似度各自的计算方式和衡量特征, 它们分别适用于不同的数据分析模型: 欧氏距离能够体现个体数值特征的绝对差异, 所以更多地用于需要从维度的数值大小中体现差异的分析, 如使用用户行为指标分析用户价值的相似度或差异; 而余弦相似度更多的是从方向上区分差异, 而对绝对的数值不敏感, 更多地用于使用用户对内容的评分来区分用户兴趣的相似度和差异, 同时修正用户间可能存在的度量标准不统一的问题 (因为余弦相似度对绝对数值不敏感)。

在实际应用中选择合适的距离度量或相似度度量, 可以完成很多的数据分析和数据挖掘的建模。

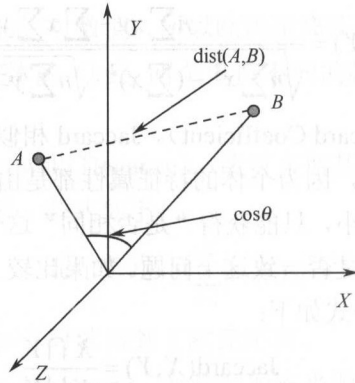


图 12.1 三维坐标系

2. 数据归一化

在计算距离和相似性时，首先要对数据进行归一化操作。

在实际使用数据时，往往需要将数据归一到一定范围以内，以方便比较或者后续处理。以下是常用的归一化方法。

(1) 线性归一化

线性归一化是最常用也是最简单的方法，这种方法比较适用于数值比较集中的情况。计算方法为 $(V - V_{\min}) / (V_{\max} - V_{\min})$ 。但采用这种方法时如果 V_{\max} 和 V_{\min} 不稳定，很容易使得归一化结果不稳定，从而导致后续使用效果也不稳定。所以实际应用中，经常用常量来替代 V_{\max} 或者 V_{\min} 。如果担心归一化结果越界，可以再加一个约束。

(2) 非线性归一化

这种方法经常用在数据分化比较严重的场合，有些数值很大，有些数值很小。

通过一些数学函数，将原始值进行映射。该方法包括 log、指数、正切等。需要根据具体的情况，确定非线性函数的曲线。

(3) 分段

可以直接将 V 的数值进行分段，每段再用其他方法进行归一。这种方法相对复杂一些，需要估计应该在哪个值进行分段，以及分段后如何进一步处理数据，不同数据之间如何进行比较等，但如果处理得好，往往可以得到比较理想的归一效果。

(4) 排序归一

有些场景对于待归一的具体值并不是很关心，更关心的是相对排序，则可以将待归一的数值排序，再进行归一。计算方法为 $1 - N_v / ALL$ ， N_v 为待归一值在所有值中的排名， ALL 为所有样本。

12.2 计算广告

12.2.1 计算广告简介

计算广告是一门新兴的多学科的交叉学科,与大型搜索、文本分析、信息检索、机器学习、分类、优化和微观经济学等诸多学科紧密融合。计算广告学是一门广告营销科学,以追求广告投放的综合收益最大化为目标,重点解决用户与广告匹配的相关性和广告的竞价模型的问题。计算广告学涉及自然语言处理、数据挖掘、竞价营销、创意设计等诸多学科。

计算广告学之所以能够兴起,主要源于互联网公司的大数据能力。大数据解决了几个计算广告学相关的关键问题:海量用户数据的挖掘、实时大数据计算(流计算)、用户与广告特征提取与匹配、语义网络的构建。

计算广告的运作系统主要包括广告算法、广告、语境、受众4个方面。根据这4个方面可将目前的计算广告形式归纳为以下3类。

1. 基于文本分析的计算广告

文本分析是近年来自然语言处理学科的一个热点研究问题,该学科一些应用研究学者相继将网页分析、文本倾向性分析、文本相似性分析、机器翻译等研究成果应用到网络广告实践中。

2. 基于用户分析的计算广告

基于用户分析的计算广告是直接寻找广告与用户的一致性。当前用户分析主要从IP、注册资料、服务器日志、Cookie、历史数据、浏览器行为等方面切入。

3. 基于用户参与的计算广告

文本和用户行为都可以通过相关算法进行兴趣相似性分析,然而图像、视频这种多媒体数据在现有的图像识别技术下,尚不能进行主题分析,因此需要人工参与。基于用户参与的计算广告系统,其主要目的在于搭建一个用户、广告主、站长的联盟平台。

12.2.2 计算广告发展阶段

计算广告参与者有以下几类。

- ① 用户:互联网用户。用户在互联网上获取内容或服务,也是广告的受众。
- ② Publisher:互联网内容或服务提供商,是互联网广告投放的媒介。用户在浏览其内

容或使用其服务的时候，相应地在该媒体上完成广告信息的接收和操作。

③ **Ad Network**：广告联盟网络，是一个连接广告主（Advertiser）和互联网媒体（Publisher）的广告系统平台，一方面为广告主提供市场营销工具甚至广告投放服务，另一方面为互联网媒体兑现部分广告的价值。

④ **Advertiser**：广告主，是营销的主体，具有投放广告到用户的商业需求并期望用户通过广告的影响成为其产品或服务的消费者。

⑤ **Ad Exchange**：一个在不同 **Ad Network** 之间实现广告与流量交换交易的平台；该平台能够在不同商业模式之间实现市场互通，进而完成广告市场的整合与利益最大化。

第一阶段：一个广告主如果需要投放互联网在线广告，他需要选择一个 **Ad Network**，并依据相关营销工具进行广告创意制作、投放计划管理和投放过程及效果监控。而事情的起点就是，把广告计划建立起来，并付款——付款购买的是什么？是 **Ad Network** 为他带来的潜在消费者——而体现就是 **Publisher** 的流量。所以，这个环节只有广告主的意愿是不够的。**Ad Network** 需要获取大量的流量位置用来投放广告——并在营销工具中告诉广告主，你的广告可以通过什么方式选定广告位置——也就是选定潜在客户。这些潜在客户的广告触受行为，才是广告主真正要花钱购买的东西。另外一个视角，就是 **Ad Network** 去建立一个广告营销市场，它需要有强大的销售力量，找到所有它需要的广告主进行广告营销，并能够获取和梳理 **Publisher** 流量位置，告诉广告主它们的营销价值值得他付款。

第二阶段：**Ad Network** 将选择合适的广告投放平台，对广告主的广告进行分析并添加到广告投放系统中。现在一个用户访问某个 **Publisher**，通过其内容或服务获取相关信息。如果当前 **Publisher** 参与了该 **Ad Network** 的广告投放活动，用户的一次 PV 将触发一次广告投放操作——系统自动将用户的部分信息（一般是有利于广告投放的信息）作为输入发送到广告投放平台，并经过一个复杂的决策过程，优选出一批广告回传至 **Publisher**，以事先设计的样式进行广告展现。此时用户就看到了互联网在线广告。

第三阶段：广告主对于“用户”的“营销消费过程”还在不同程度地延续。如果是 CPM——千次展现付费——则广告浏览本身已经形成费用；系统需要将这个浏览信息记录，并启动后续广告费用及报表相关的操作。如果是 CPC——点击付费——则还需要看用户是否进一步点击了广告：如果用户没有点击，则广告展现分文不取；如果用户点击了，则需要将点击信息记录，并启动后续广告费用及报表相关操作。当然，还有其他更为复杂的商业模式。此时广告主的费用已经被扣减了，**Publisher** 也应该拿到了自己应得的广告费用分成部分。

传统的 **Ad Network** 交互流程如图 12.2 所示。

目前主要模式为 RTB 模式，如图 12.3 所示。



图 12.2 传统的 Ad Network 交互流程



图 12.3 RTB 模式

下面从不同角度看 RTB 与传统 Ad Network 的区别。

从 Ad Network（或 RTB 中的 DSP）的角度：

- ① 原来从媒体来的每一个广告请求，通常都能获得展现机会，现在不一定了。
- ② 原来用户没点击，Ad Network 不用出钱（因为大多采用 CPC 结算），现在则需要出钱（因为采用 CPM 结算）。原来稳赚不赔，现在弄不好还亏本。
- ③ 媒体垄断优势没了，竞争者变多了，广告主可能变少了。
- ④ 不用去和各家媒体一一谈合作了。

从广告主的角度：

- ① DSP 比 Ad Network 更能代表自己的利益，可以提出更多个性化的投放需求。
- ② 更容易在投放中定制化地使用广告主自己的数据和第三方数据。

③ DSP 的优化效果可能比 Ad Network 更好,但也可能更差。

从媒体主的角度来看,收益理论上会增加,但在早期市场中竞价不激烈时,收益可能降低。

12.2.3 计算广告相关算法

1. DSP 的工作流程

在讲解算法之前,先大致介绍一下 DSP 的工作流程。

(1) 追踪网民行为 (Behavior Tracking)

① Action Data (广告主的数据)。DMP 公司在广告主的网站上埋点 (通常是放上一个 1×1 的不可见像素),这样当网民访问广告主的网站时,DMP 公司会得到该信息。在广告主授权下,DMP 公司把该数据传给 DSP。

② Mapping Data (媒体的数据)。DSP 还会和第三方网站合作 (如新浪、腾讯),在他们的网站上也埋点,或者向 DMP 公司购买网民行为数据,这样就可以追踪到网民在这些网站上的行为。网民在每个网站上留下的 Cookie 不一样,需要做 Cookie Mapping。

(2) 受众选择 (Audience Selection)

① 离线计算每个 Campaign 的目标投放用户集。

② 广告主 (或账户操作人员) 可以通过配置来管理这些目标投放用户集。

(3) 进行实时竞价 (Bidding)

① 当 Ad Exchange 把请求发过来的时候,DSP 会得到以下信息:

- 当前广告位的信息;
- 当前用户的 Cookie 和基本信息。

② DSP 需要在 100ms 内,根据对当前用户的理解,并且考虑当前广告位,依据自己的 Bidding 算法来决定:

- 是否要对这次展现机会进行竞价;
- 投放哪个 Campaign 的广告;
- 出价是多少。

(4) 展现广告

如果出价最高,赢得了展现机会,则 DSP 返回创意,网民就会在该广告位看到该创意 (图片、文字、Flash)。

(5) 追踪转化

① Ad Exchange 向 DSP 反馈该 DSP 竞价成功的展现是否造成点击或转化。

② 根据这些数据统计点击率 (CTR)、转化率 (CVR)、每个转化平均成本 (CPA) 等各种指标,汇总成报表展示给广告主。

2. 相关算法

上述过程中的算法,大致描述如下。

(1) 目标用户选择 (Audience Selection)

找到每个 Campaign 的目标投放用户集。

① 基于标签的做法 (与 Ad Network 差异不大)。

- DSP 对所有能追踪到的网民, 根据其行为为每个网民打上各种标签 (User Profiling)。
- 广告主 (或账户操作员) 对每个 Campaign 选择一系列标签, 从而确定自己的目标投放用户集。

② 基于重定向的做法。

重定向的方式很多, 如 KT 重定向、Cookie 重定向。Cookie 重定向就是记录曾经访问过广告主网站的 Cookie, 然后广告主只对这些 Cookie 进行投放。

③ 基于 Look-alike 模型的做法 (以 M6D 的做法为例)。

对每个 Campaign, 建立模型预估用户发生转化的概率 $P(c|u)$ 。

正例是在广告主网站发生转化的用户, 反之为负例, $P(c|u)$ 由两级模型来构建。

根据每个用户的 $P(c|u)$ 将用户划分到不同的 Segments。

不同 Segments 的 $P(c|u)$ 范围不一样, 平均每个 Campaign 有 10~50 个 Segments。广告主根据自己的需求, 决定开启或关闭某些 Segments。

(2) 出价 (Bidding Algorithm)

当 Ad Exchange 发送竞价请求时, 携带了网民 Cookie 信息和广告位信息。

① 检索: DSP 先根据 Cookie 找到所有目标投放用户集中包含该 Cookie 的 Campaign。

② 过滤: 筛掉那些达到预算限制的 Campaign, 以及对当前用户达到展现次数上限的 Campaign。

③ 出价: 对每个 Campaign 计算出一个出价。

④ 内部竞价: 选择出价最高的 Campaign, 并把出价返回给 Ad Exchange。

以上过程需要在 100ms 内 (或更短) 完成。

(3) 调整出价 (Bid Adjustment)

在线上生产环境中进行实际竞价时, 通常需要对竞价模型的参数做调整。

原因:

- 线上的数据分布与线下用的训练数据的分布不一样, 需要对参数做调整;
- 线上的环境是动态变化的, 参数也应随之变化。

常见的算法, 有以下两种。

① 预测 (Forecasting)。

预测对象:

- 流量, 即预估未来的流量大小;
- 在不同的出价下, 能赢得展现的概率分布。体现竞争对手的出价情况。

预测范围:

- 全流量下的预估;
- 不同定向条件下的预估。

② 反馈控制。

以消费控制为例，计算公式为：

$$\alpha(t+1) = \alpha(t) \exp(\lambda(\frac{\text{spend}(t-1,t)}{\text{budget}} - \frac{1}{T}))$$

式中， λ 为参数。

上式控制每个时间间隔的消费一致，但实际应用中通常不是一致的：

$$\alpha(t+1) = \alpha(t) \exp(\lambda(\frac{\text{spend}(t-1,t)}{\text{budget}} - f(t,T)))$$

式中， $f(t,T)$ 为 $t-1$ 到 t 时间段的消费控制目标。

算法汇总见表 12.2。

表 12.2 算法汇总

算法名称	作用	常用算法
目标用户选择 (Audience Selection)	找到每个 Campaign 的目标投放用户集	① 基于标签的做法 (与 Ad Network 差异不大) ② 基于重定向的做法 ③ 基于 Look-alike 模型的做法
出价 (Bidding Algorithm)	实时竞价过程	基于价值的出价
调整出价 (Bid Adjustment)	实时调整出价策略	① 预测 ② 反馈控制

12.2.4 计算广告与大数据

广告正在从 Ad Network 向 Ad Exchange 转变，大数据挖掘和分析正在发挥重要作用。在国外，DMP（数据管理平台）已快速发展起来，能够通过整合不同数据源，进行受众分析，帮助判断目标受众和单个受众价格，让广告与用户需求能更精准地对接。比如从定向方式来看，出现了关键词重定向，当用户搜索完关键词“DSP”后浏览其他网站时，就会出现如图 12.4 所示的推荐。



图 12.4 推荐示例

12.3.1 社交网络中大数据挖掘的应用场景

1. 意见传播、动态网络影响力传播模型分析

这是社交网络分析的典型应用之一，主要分析相关主题图结构数据中的“意见领袖”、“结构洞”（即跨越不同社群子网络的桥接节点）、“动态网络影响力传播模型”等问题。

2. 某领域专家发现和排名

基于某个学术主题或学术会议，在相关论文的合作者构成的图数据中，找到最有影响力的专家，分析专家影响力的排名，并图形化呈现专家与专家之间、专家与研究课题之间，以及研究课题与相关学术会议之间的关系，便于人们直观地发现某领域内专家的排名顺序和相互之间的关系。

3. 社交关系分析

按照社交网络的六度空间理论，每两个人的关系一般只需要通过 6 个中间人就可以建立。所以在社交媒体中，人们之间的关系基本都可以组成网络结构。社交关系分析，最典型的应用案例就是通过用户的电话记录或者邮件记录，分析其中哪些人是家人，哪些人是同事等。

4. 相关主题的历史和趋势分析

针对某个主题，其表达方式在不同的时间会有所不同，还会有一些相关的子主题。这些不同的表达方式或子主题就组成了针对某个主题的演进关系图。

5. 基于地理位置的某领域专家分布分析

基于全球地图，查看某个领域全球顶尖专家的分布，进一步看出全球各个地区在该领域研究力量的分布，如一流的专家有哪些，分别聚集在哪个地区。

6. 知识图谱的构建

知识图谱是谷歌、百度、雅虎搜索等知名搜索引擎近几年新发展的技术，其核心是为用户提供查询信息与相关知识的关系。对用户而言，直接通过图示的方法展现密切关联的信息，比仅提供网页链接，价值要大很多。

12.3.2 社交网络大数据挖掘核心算法模型

1. 社群发现

在社交网络中，有相似特征的成员会自动形成一些社群。通过计算自动地发现这些社

群，可以帮助人们识别一个个有相似特征的群体，也就可以针对这些人群的特征分门别类地做各种应用。这里使用的是相关主题的图节点聚类方法，主要使用 FCM 算法，即基于模糊集的均值聚类算法。

2. 专家排名

学术论文中，每篇文章的合作者可以构成一个网络，而且这个网络是基于该论文主题的专家网络。所以，根据专家基本信息给出初始分数，针对某个主题的很多论文的专家网络关系，通过 ACT (Author Conference Topic) 模型就可以迭代计算出每个节点 (专家) 的排名。这也被称为基于传播的算法。

3. 社交网络节点影响力的算法

通过 Topical Affinity Propagation (TAP) 模型，可以基于某个主题在社交网络构建影响力模型。这个模型基于因子图 (Factor Graph)，因此又被称为 Topical Factor Graph (TFG) 模型。

4. 对于网络中节点关系的自动标注

在很多情况下，各种不同网络中的数据关系是未知的，或者只有小部分数据有关系标注 (Label)，大部分数据是没有关系标注的。这就需要一些半自动的算法进行关系标注，半监督 (Semi-supervised) 算法应运而生。

5. 图模型算法的并行化分布式计算

图数据的挖掘往往涉及海量数据，而且算法比较复杂，有效地进行并行化分布式算法处理是一个重点。

12.3.3 图计算框架

1. Google Pregel

Pregel 是 Google 继 MapReduce 之后提出的又一个计算模型，与 MapReduce 的离线批处理模式不同，它主要用于图的计算。该模型的核心思想源于 Leslie Valiant 于 20 世纪 80 年代提出的 BSP 计算模型。图计算涉及在相同数据上的不断更新以及大量的消息传递，如果采用 MapReduce 去实现，会产生大量不必要的序列化和反序列化开销。在传统的高性能计算领域通常会借助 MPI 来完成，但是 MPI 本身只是提供了一系列的通信接口，开发难度较高，同时对于 Google 由普通 PC 构建的集群来说，MPI 的容错性不够。而现有的一些图计算系统也并不适应 Google 的场景。

Pregel 计算由一系列的超级步组成，在每个超级步内，框架会调用每个顶点的用户自定义函数，该函数定义了在该超级步内需要进行的计算，该函数可以读入前一个超级步发送给它的消息，并产生下一轮迭代的消息给其他顶点，同时会修改顶点及其边的状态。

2. Apache Hama

Apache Hama 支持本地、伪分布式、分布式模式。Hama 本身与 Hadoop 十分类似，只是具有自己的通信和同步机制，利用 Hadoop RPC 进行节点间通信，借助于 ZooKeeper 进行同步，通过对消息进行收集和捆绑发送来降低网络开销和竞争。

3. GoldenOrb

GoldenOrb 的关键构建块由 Vertex、VertexBuilder、VertexWriter 和 OrbRunner 构成。其中 Vertex 由 id、value 及边组成，代表图中的一个节点。VertexBuilder 负责 Vertex 的构建，通过 buildVertex 函数创建 Vertex 对象。VertexWriter 则用来为 Vertex 对象创建相关的 OrbContext 对象。OrbRunner 则用于启动 job，主要负责解析命令行参数、初始化、连接 ZooKeeper 以及设置好 OrbConfiguration。

4. 微软 Trinity

Trinity 是微软开发的一套图计算平台，包含一个建立在分布式内存云平台上的图数据库及一个计算框架。通过一个纯内存的 key/value 存储数据库实现快速访问。

Trinity 的一个基本存储单元称为一个 cell，每个 cell 通过一个全局唯一的 id 标识，该 id 是一个 64bit 整数，支持用户通过这个 id 进行随机访问。从底层 key/value 的存储角度来看，key 就是 cell-id，value 是一个任意长度的字符串。

5. Spark Graphx

Spark 主要用来解决 MapReduce 所不擅长的两类计算：迭代计算和交互式分析。核心在于将数据存在内存中，避免重复的 load。采用 Scala 语言实现，提供了类似于 DryadLINQ 的函数式编程接口。Spark 为并行编程主要提供了两个抽象：RDD 和并行操作。此外它还提供了两种类型的共享变量支持：广播变量和累加器。

6. 豆瓣 Dpark

Dpark 是豆瓣刚开源的集群计算框架，是 Spark 的 Python Clone 版本，类似于 MapReduce，但是比其更灵活，可以用 Python 非常方便地进行分布式计算，并且提供了更多的功能以便更好地进行迭代式计算。Dpark 的计算模型基于两个中心思想：对分布式数据集的并行计算，以及一些有限的可以在计算过程中从不同机器访问的共享变量类型。Dpark 具有一个很重要的特性：分布式的数据集可以在多个不同的并行循环当中被重复利用。这个特性将其与其他数据流形式的框架如 Hadoop 和 Dryad 区分开来。

Dpark 与 Spark 的区别：Spark 中使用一个线程运行一个任务，但是 Dpark 受 Python 中 GIL 的影响，选择使用一个进程来运行一个任务；Spark 支持 Hadoop 的文件系统接口，Dpark 只支持 POSIX 文件接口。

7. CMU 的 GraphLab

GraphLab 是 CMU（卡耐基·梅隆大学）开发的一个以 Vertex 为计算单元的大规模图

处理系统，是继 Google 的 Pregel 之后第一个开源的大规模图处理系统，它解决了传统 MapReduce 框架对于机器学习应用的处理中最突出的两个问题（频繁迭代计算和大量节点通信）引起的计算效率的问题，与 Haloop、Twister 等基于 MapReduce 批量处理不同的是，它以 Vertex 为计算单元，并将机器学习抽象成 GAS（Gather, Apply, Scatter）三个步骤，然后按该抽象模型设计实现算法，事实已经证明该框架对于机器学习这一类跟图处理关系紧密的应用有很好的效果。

8. 开源 Giraph

Apache Giraph 是一个高扩展性的交互图形处理系统。它被用于 Facebook 的社交图谱分析，用来形成网络连接。Giraph 是 Pregel 的开源版本。Giraph 除了基本的 Pregel 模型外，还添加了许多特性，包括 master computation, shared aggregators, edge-oriented input, out-of-core computation 等。

9. Neo4j

Neo4j 是一个面向网络的数据库，也就是说，它是一个嵌入式的、基于磁盘的、具备完全的事务特性的 Java 持久化引擎，但是它将结构化数据存储在网上而不是表中。网络（从数学角度叫做图）是一个灵活的数据结构，可以应用更加敏捷和快速的开发模式。

12.3.4 大数据在社交网络中的应用案例

社交网络中有很多的应用场景，主要集中在广告营销、产品服务和用户管理三个层面。本节将以好友推荐为例来介绍大数据在社交网络中的应用。

其实，社交网站上的各个用户以及用户之间的相互关注可以抽象为一个图。本节以图 12.6 为例。

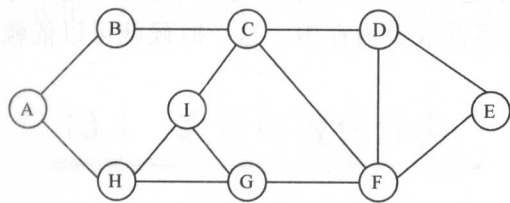


图 12.6 用户之间关系抽象图

顶点 A~I 分别表示社交网站的用户，两顶点之间的边表示两顶点代表的用户之间相互关注。那么如何根据用户之间相互关注所构成的图来向每个用户推荐好友呢？可能大家都听说过六度人脉的说法，所谓六度人脉是指地球上所有的人都可以通过六层以内的熟人链和任何其他人联系起来。这个理论在社交网络中同样成立。

以图 12.7 为例，首先我们假设如果两用户之间相互关注，那么即可认为他们认识，或者说是现实中的好友。假设现在需要向用户 I 推荐好友，我们发现用户 I 的好友有 H、G、C。其中 H 的好友还有 A，G 的好友还有 F，C 的好友还有 B、F。那么用户 I、H、G、C、A、B、F 极有可能是同一个圈子里的人。我们应该把用户 A、B、F 推荐给用户 I 认识。进一步分析，用户 F 跟 I 的两位好友 C、G 是好友，而用户 A、B 都分别只跟 I 的一位好友是

好友，那么相对于 A、B 来说，F 当然更应该推荐给用户 I 认识。

在上面的分析中，我们使用了用户 I 的二度人脉作为他的推荐好友，而且我们对用户 I 的每个二度人脉进行了投票处理，选举出最优推荐。

所谓的 N 度人脉，其实就是图算法里面的宽度优先搜索。宽度优先搜索的主要思想是 From Center To Outer。以用户 I 为起点，在相互关注所构成的图上往外不退回地走 N 步所能到的顶点，就是用户 I 的 N 度好友。图 12.7 显示了用 Python 语言写的 N 度人脉的算法，可以输出某个用户的 N 度好友（<https://github.com/intergret/snippet/blob/master/AlgorithmsInGraph.py>）。

```

24 def NDegreeNode(self, start, n):
25     pathFromStart = {}
26     pathFromStart[start] = [start]
27     pathLenFromStart = {}
28     pathLenFromStart[start] = 0
29     todoList = [start]
30     current = todoList.pop(0)
31     neighbor = self.Link[current]:
32     neighbor = pathFromStart:
33     pathFromStart[neighbor] = pathFromStart[current] + [neighbor]
34     pathLenFromStart[neighbor] = pathLenFromStart[current] + 1
35     pathLenFromStart[neighbor] = n-1:
36     todoList.append(neighbor)
37
38 for node in pathFromStart.keys():
39     len(pathFromStart[node]) = n-1:
40     pathFromStart[node]
41
42 return pathFromStart

```

图 12.7 代码示例

上述算法与宽度优先搜索的不同之处如下：

- ① 宽度优先搜索的是起始顶点可达的所有顶点， N 度人脉不需要，它只需要向外走 N 步，走到 N 步的顶点处便停止，不需要再往外走了。
- ② 走过 N 步之后，结果中包含起始顶点往外走 1, 2, ..., $N-1$ 步所能到达的所有顶点，返回结果之前需要将这些点删除。
- ③ 变量 `pathLenFromStart` 记录这 N 步具体的走法。

当然上述算法看似可行，其实在实际中并不适用。社交网站上的用户量至少是千万级别的，不可能把所有用户之间相互关注的关系图放进内存中，这个时候就可以依赖 Hadoop 了。

12.4 练习题

1. 用协同过滤推荐算法设计一个推荐书籍的实例。
2. 计算广告的形式有哪些？
3. 描述一下计算广告的相关技术。
4. 描述大数据在社交网络中的典型应用场景。
5. 描述 Graphx 图计算框架的架构。

第13章

大数据展示与交互技术

一幅图胜过千言万语。人类从外界获得的信息约有 80%以上来自视觉系统。当大数据以直观的可视化的图形形式展示在分析者面前时，分析者往往能够一眼洞悉数据背后隐藏的信息并将其转化为知识以及智慧。所以说，在大数据技术体系中，数据展示与交互虽不是核心，但也至关重要。数据处理的最终目的是使人们更好地利用数据，选择恰当的、生动直观的展示方式能够帮助人们更好地理解数据的内涵和关联关系，也能够更有效地解释和运用数据，从而为生产、运营、规划提供决策支持，发挥出大数据的作用。随着技术的发展，大数据的展现方式也发生了巨大变化，除了传统的报表、图形之外，结合现代化的可视化工具及人机交互手段，智能化、实时化、多维度的数据展示与交互时代已经来临。

13.1 数据可视化分类

随着数据仓库技术、网络技术、电子商务技术的发展，可视化技术涵盖了更广泛的内容，并进一步提出了数据可视化的概念。所谓数据可视化（Data Visualization）是对大型数据库或数据仓库中的数据的可视化，它是可视化技术在非空间数据领域的应用，使人们不再局限于通过关系数据表来观察和分析数据信息，还能以更直观的方式看到数据及其结构关系。数据可视化技术的基本思想是将数据库中每一个数据项作为单个图元元素表示，大量的数据集构成数据图像，同时将数据的各个属性值以多维数据的形式表示，可以从不同的维度观察数据，从而对数据进行更深入的观察和分析。

根据不同的划分维度，数据可视化分类的方法有很多。本节主要介绍两种分类方法：按照可视化展示的内容分类和按照原始数据类型分类。

13.1.1 按照展示内容进行划分

从数据展示的角度来看，可视化技术主要是针对数据的结构、功能、关联关系、发展趋势等几个方面进行展示。

1. 结构可视化

这主要用来反映数据的内在组织结构，比如构成数据的元素、部件以及构成关系等，在医学和生物学领域应用较多。典型的例子是生物蛋白质结构可视化，如图 13.1 所示。

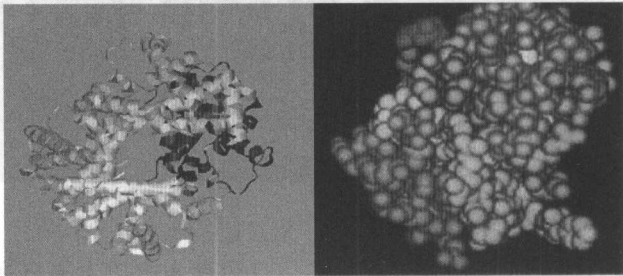


图 13.1 蛋白质结构可视化

2. 功能可视化

这是对数据所对应的功能可视化描述，比如汽车发动机的运转状态，就可以通过对发动机进行 3D 建模，形成一段动画来清晰地进行展示。图 13.2 是设计师 Jing Zhang 设计的 iPhone 和数码相机的信息图，这些图片不但准确解释了设备的工作原理，还极大地激发了读者的想象力。

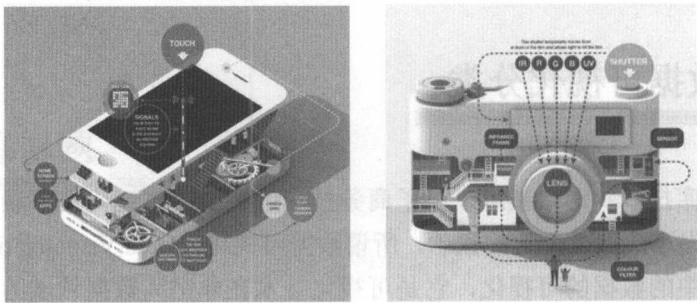


图 13.2 iPhone 和数码相机的信息图

3. 关联关系可视化

大数据可视化在很大程度上都是反映数据之间的关联关系，比如层级关系、对比关系

之类的社交图谱。

俄罗斯工程师 Ruslan Enikeev 根据 2011 年年底的数据,将 196 个国家的 35 万个网站数据整合起来,把每个网站都看成一个“星球”,并根据 200 多万个网站间的链接将这些“星球”通过关系链联系起来,形成了互联网“星球”图,如图 13.3 所示。每一个“星球”的大小根据其网站流量来决定,而星球之间的距离则根据链接出现的频率、强度和用户跳转时创建的链接决定。这些星球有恒星、行星甚至卫星,每一个星球都有其特定的星系。

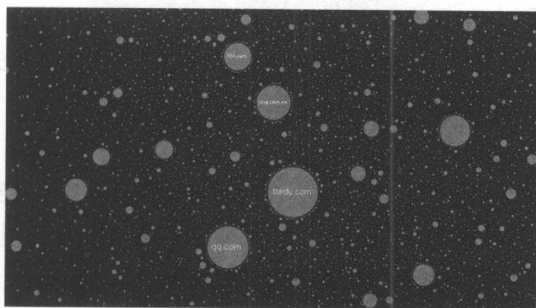


图 13.3 互联网“星球”图

4. 发展趋势可视化

这是对数据发展的走势、预测等进行可视化的一种方式。谷歌的设计人员认为,人们输入的搜索关键词代表了他们的即时需要,反映出的是用户需求。为了使用户搜索与流感爆发建立关联,设计人员编入了一系列的流感关键词,包括温度计、咳嗽、发烧、肌肉疼痛、胸闷等,只要用户输入这些关键词,系统就会展开跟踪分析,创建地区流感图表和流感地图。为验证谷歌“流感趋势”预警系统的正确性,谷歌多次把测试结果与联邦疾病控制和预防中心的报告做对比,证实两者结论存在很大相关性,而且谷歌能够比联邦疾病控制和预防中心提前 7~14 天准确预测流感的暴发。

13.1.2 按照数据类型进行划分

随着大数据的兴起与发展,互联网、社交网络、地理信息系统、企业商业智能、社会公共服务等主流应用领域逐渐催生了几种特征鲜明的信息类型,主要包括文本、网络图、时空及多维数据等。这些与大数据密切相关的信息类型与多维数据分类模型交叉融合,将成为大数据可视化的主要研究领域。

1. 文本可视化

文本信息是大数据时代非结构化数据类型的典型代表,是互联网中最主要的信息类型,也是物联网各种传感器采集后生成的主要信息类型,人们日常工作和生活中接触最多的电子文档也以文本形式存在。但是随着信息量日益增大,相互关系也越来越复杂,人们处理和理解这些信息的难度日益增大,传统的文本分析技术无法满足人们利用浏览及筛选等方

图用于呈现文本中命名实体的从属关系、并列关系等；径向空间填充用于呈现词语的层次关系或词语在 Wordnet 中的上下位关系及词频，比如 DocuBurst 以放射状层次圆环的形式展示文本结构（图 13.5）。

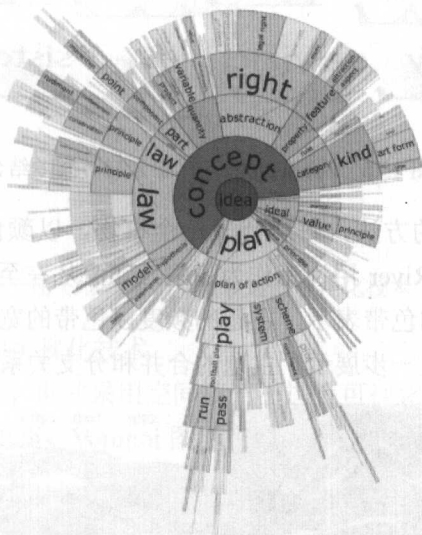


图 13.5 DocuBurst 示例

基于文本外在关系的可视化反映的是文本间的引用关系、网页的超链接关系等直接关系，以及主题相似性等潜在关系（一般基于聚类算法，用来呈现主题分布，并展示与特定主题相关的关键词，主要应用于信息检索、主题检测、话题演变等方面）。可视化形式主要有网络图、FP-tree、标签云改造几种。其中网络图主要用来展示对文本集的引用关系，网络节点代表文本，有向线代表引用关系；FP-tree 用来展现文献共引关系，可以比 CiteSpace 这种传统网络图可视化方案呈现更为细致的信息，便于学术领域研究；标签云改造则可以呈现由 Jaccard 系数计算出的聚类结果，同行同主题，相邻行主题相似。

（3）包含时间关系的可视化

文本的形成和变化过程与时间属性密切相关，因此，将动态变化的文本中时间相关的模式与规律进行可视化展示是文本可视化的重要内容。包含时间关系的可视化方案主要思想是通过时间信息提供文本内容变化等数据规律信息。包含时间关系的可视化的具体实现包括引入时间轴、信息按时间顺序排列、标签云与时间结合、叠式图（stacked graph）等几种形式。

标签云与时间结合具有以下几种表现形式：一是在词语下引入折线图，表示词语使用频度的变化，如图 13.6 所示；二是在标签云上标上不同颜色和图形；三是使用时间折线图或时间点标签云，折线图上值越大表示此时的标签云标签越多。

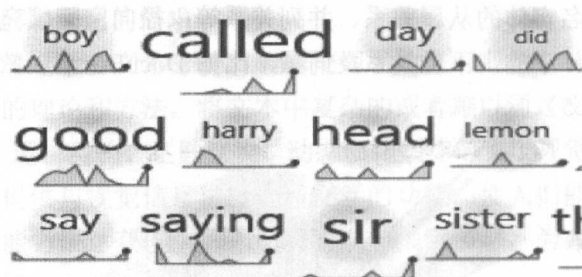


图 13.6 SparkCloud: 标签云与时间轴结合

叠式图使用分层叠加的方法，每层代表一个事物，以颜色区分，粗细代表频度，如图 13.7 所示。其中 ThemeRiver 用河流作为隐喻，河流从左至右的流淌代表时间序列，将文本中的主题用不同颜色的色带表示，主题的频度以色带的宽窄表示。基于河流隐喻，研究者又提出了 TextFlow，进一步展示了主题的合并和分支关系以及演变进程。

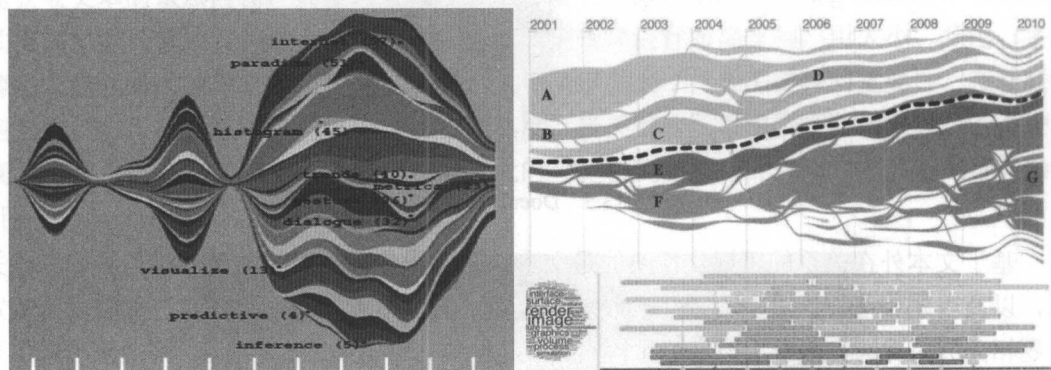


图 13.7 ThemeRiver 与 TextFlow

2. 网络图可视化

网络关联关系是大数据中最常见的关系，例如互联网与社交网络。层次结构数据也属于网络信息的一种特殊情况。基于网络节点和连接的拓扑关系，直观地展示网络中潜在的模式关系（例如节点或边的聚集性），是网络可视化的主要内容之一。对于具有海量节点和边的大规模网络，如何在有限的屏幕空间中进行可视化，是大数据时代面临的难点和重点。除了对静态的网络拓扑关系进行可视化，大数据相关的网络往往具有动态演化性，因此，如何对动态网络的特征进行可视化，也是不可或缺的研究内容。

网络图可视化技术有很多类型，主要可分为经典的基于节点和边的可视化技术、基于空间填充法的可视化技术和基于图简化方法的可视化技术。

(1) 基于节点和边的可视化技术

Herman 等人综述了图可视化的基本方法和技术，如图 13.8 所示。图中主要展示了具有层次特征的图可视化的典型技术，例如 H 状树、圆锥树、气球图、放射图、三维放射图、双曲树等。

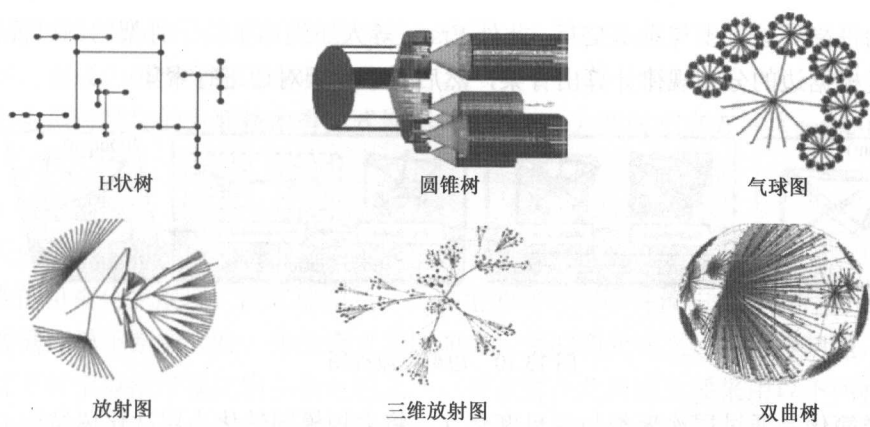


图 13.8 基于节点和边的可视化技术

(2) 基于空间填充法的可视化技术

对于具有层次特征的图，也常采用空间填充法进行可视化，如树图技术及其改进技术。如图 13.9 所示是基于矩形填充、Voronoi 图填充、嵌套圆填充的树可视化技术的示例。

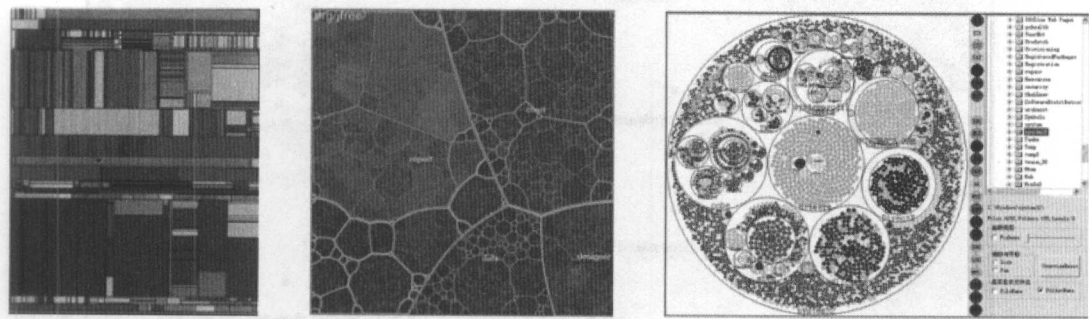


图 13.9 基于空间填充法的可视化技术

这些图可视化技术的特点是直观表达了图节点之间的关系，可以比较精确、直观地显示数据节点的基本内容和相互关系，在数据规模较小时，在显示界面像素允许范围内可以较好地实现数据可视化（例如百万以内）；但数据规模较大时，海量节点和关系密集地进行填充，就会导致节点和边的覆盖重叠，使得可视化效果大打折扣。因此面临大数据中的海量节点和边，需要对这些方法进行相应的改进。

(3) 基于图简化方法的可视化技术

为应对大规模网络中海量数据节点和边在传统空间填充法中出现的重叠覆盖问题，人们提出了图简化（graph simplification）方法，具体有以下两类。

第一类简化是对边进行聚集处理，例如基于边聚集（edge clustering）的方法，使得复杂网络可视化效果更为清晰。如图 13.10 所示，基于边聚集的大规模密集图可视化技术，通过一个 Control Mesh 引导边缘聚集的过程，并且可以根据不同的图模式自动或者手动生成 Control Mesh 的不同层次的细节参数，用户还可以通过一些先进的可视化技术如颜色和

对与时间和空间密切相关的模式及规律进行展示。大数据环境下时空数据的高维性、实时性等特点，也成为时空数据可视化的重点。

时空数据可视化的主要技术有流式地图（flow map）和时空立方体（space-time cube）两种。

（1）流式地图

要反映信息对象随时间进展对应的空间位置所发生的变化，通常需要通过信息对象的特定属性的可视化来展现。流式地图是一种典型的将时间事件流与地图进行融合的方法。

当数据规模不断增大时，传统流式地图面临大量的图元交叉、覆盖等问题，这也是大数据环境下时空数据可视化的主要问题之一。目前解决此问题主要采用以下两种方法：一是借鉴网络图可视化中的边聚集方法对流式地图进行抽象和聚集；二是采用基于密度计算的方法对时间事件流进行融合处理，图 13.12 是结合了密度图技术的流式地图示例。

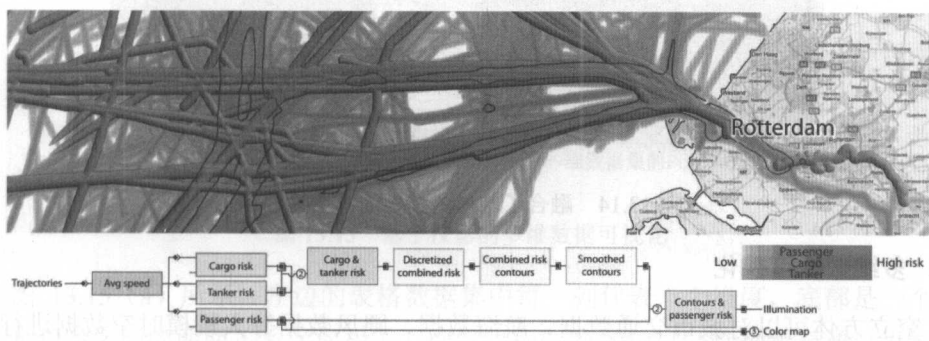


图 13.12 结合了密度图技术的流式地图示例

（2）时空立方体

时空立方体就是以三维方式对时间、空间和事件进行描述，通过立体模型直观地展现出来（图 13.13）。

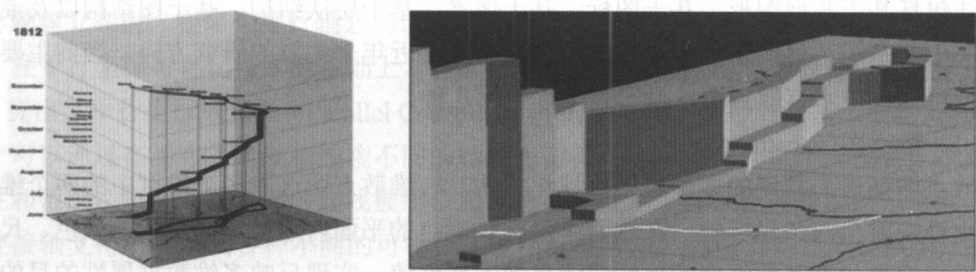


图 13.13 时空立方体示例

时空立方体同样面临着大规模数据造成的密集杂乱问题。解决这个问题的第一种思路是结合散点图和密度图对时空立方体进行优化，第二种思路是对二维和三维进行融合。Tominski 等人引入了堆积图（stack graph），这个方法的核心是 2D/3D 混合显示思想，2D

地图作为参考空间上下文，事件轨迹可视化为带颜色编码的属性值堆叠的 3D 轨迹线，实现了在时空立方体中拓展多维属性显示空间的目标，如图 13.14 所示。

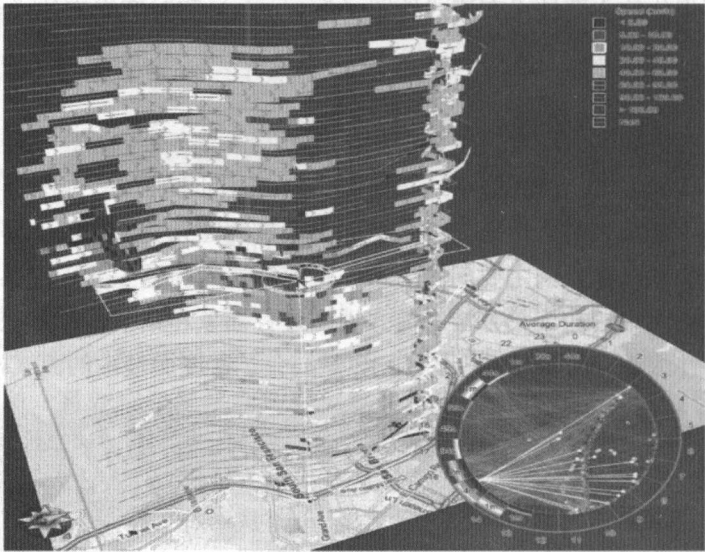


图 13.14 融合了堆积图技术的时空立方体

4. 多维数据可视化

时空立方体可以对城市交通数据、航海数据、飓风数据等大规模时空数据进行展现。但是当时空信息对象属性的维度较多时，三维立方体同样不能完美展示数据信息，这就需要进行多维数据的可视化。

多维数据指的是具有多个维度属性的数据变量，广泛存在于基于传统关系数据库以及数据仓库的应用中，例如企业信息系统以及商业智能系统。多维数据分析的目标是探索多维数据项的分布规律和模式，并揭示不同维度属性之间的隐含关系。多维数据可视化的基本方法包括基于几何图形、基于图标、基于像素、基于层次结构、基于图结构以及混合方法。其中，基于几何图形的多维数据可视化方法是近年来主要的研究方向。当前主要的多维数据可视化技术有散点图、投影和平行坐标几种。

(1) 散点图 (scatter plot)

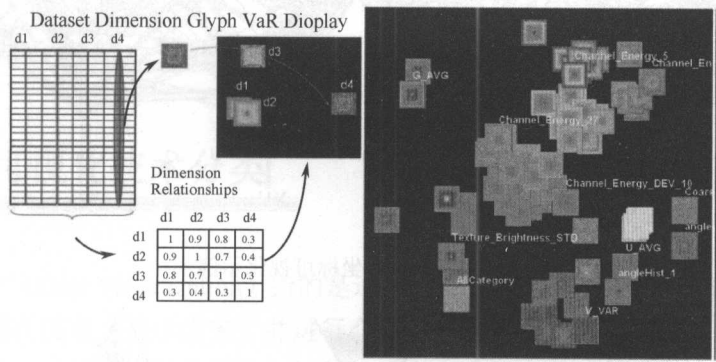
散点图是最为常用的多维数据可视化方法。二维散点图选取多个维度中的两个维度属性值集合映射至两个坐标轴，在这两个坐标轴确定的平面内通过不同形状、颜色、尺寸等的图形标记来代表其他维度的连续或者离散的属性值，实现反映多维数据属性的目的。由于二维散点图能够展示的维度十分有限，研究者将其扩展到三维空间，也就是三维散点图。这种技术通过可旋转的 scatter plot 方块 (dice) 扩展了可映射维度的数目。三维条件下，散点图场景通过三维动画旋转形式进行转换，用户可以通过设置数据集中的边界条件反复迭代构建查询，从不同视角使属性查询变得越来越完善。

散点图适合对有限数目的较为重要的维度进行可视化，通常不适于需要对所有维度同

时进行展示的情况。

(2) 投影 (projection)

为了解决高维数据集造成的可视化显示混乱和交互响应时间过长的问題，人们提出了投影的方法。Jing Yang 等人提出的 Value and Relation (VaR) 方法就是其中之一，它允许用户高效地显示几百个维度的大型数据集 (图 13.15)。



(a) 一个4维数据集对应的VaR显示 (b) 一个89维数据集的Pixel MDS VaR显示

图 13.15 基于投影的多维数据可视化

如图 13.15 (a) 所示，左边的表格数据集中每一列代表一个维度，底部是一个矩阵，代表的是记录之间的两两关系（如相关）维度。VaR 将各维度属性列集合通过投影函数映射到一个方块形图形标记中，并根据维度之间的关联度对各个小方块进行布局。基于投影的多维数据可视化方法一方面反映了维度属性值的分布规律，另一方面直观展示了多维度之间的语义关系。

(3) 平行坐标 (parallel coordinates)

平行坐标是当前研究和应用最为广泛的一种多维数据可视化技术，这种方法将维度与坐标轴建立映射，在多个平行轴之间以直线或曲线映射表示多维信息，如图 13.16 所示。

在平行坐标可视化技术的基础上，有人将平行坐标与散点图等其他可视化技术进行集成，提出了平行坐标散点图 (Parallel Coordinate Plots, PCP)。这种方法基于灵活的连接轴，用户可以通过在画布上绘制和连接不同的轴线来定义一个可视化。每个轴都有一个相关的属性和范围，每一对轴之间的连线被用来通过散点图或者平行坐标样式图显示数据。灵活的连接轴支持用户定义各种不同的可视化，包括标准方法，如散点图矩阵、PCP、雷达图表等；也包括一些新的方法，如 Hyperboxes, Time Wheels、Many-to-Many PCP (图 13.17) 等。此外，Geng 等人建立了一种具有角度的柱状图平行坐标，支持用户根据密度和角度进行多维分析。

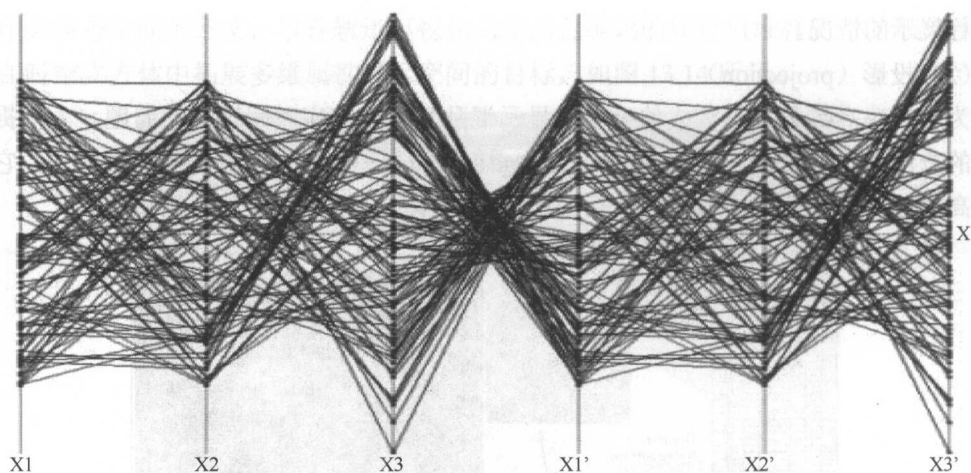


图 13.16 平行坐标可视化示例

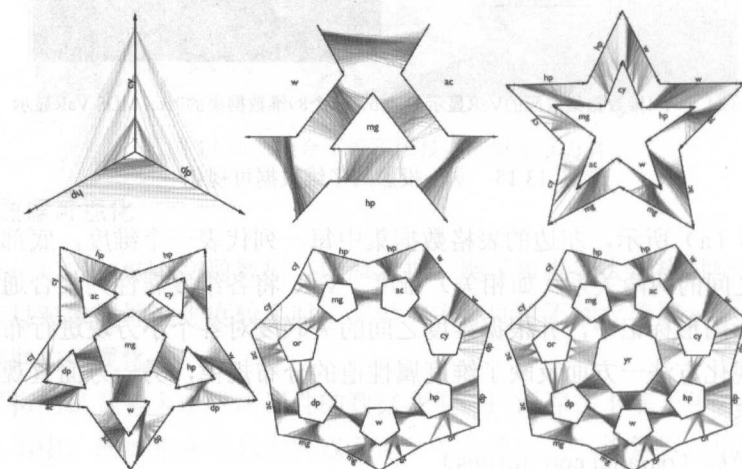
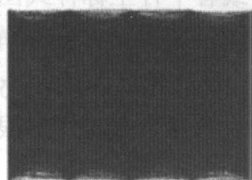
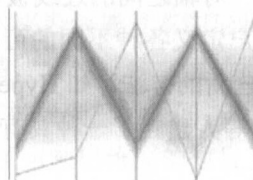


图 13.17 Many-to-Many PCP

随着数据规模不断增大，平行坐标技术同样面临大规模数据项造成的线条密集与重叠覆盖问题。类似于图简化中的边聚集技术，根据线条聚集特征对平行坐标图进行简化，形成聚簇可视化效果，为这一问题提供了有效的解决方法，如图 13.18 所示。

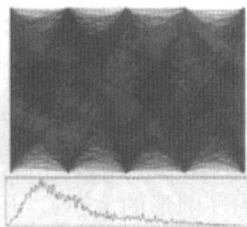


(a) 原始数据点

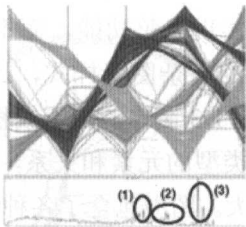


(b) 分层平行坐标结果

图 13.18 在具有 5 个变量和 7736 个数据项的数据集上的实验效果



(c) 应用转换函数之后进行视觉聚类



(d) 视觉聚类后应用转换函数

图 13.18 在具有 5 个变量和 7736 个数据项的数据集上的实验效果 (续)

13.2 可视化技术分类

数据可视化 (Data Visualization) 指的是运用计算机图形学和图像处理技术, 将数据转换为图形或图像在屏幕上显示出来, 并进行交互处理的理论、方法和技术。它涉及计算机图形学、图像处理、计算机辅助设计、计算机视觉及人机交互技术等多个领域。传统的可视化交互通常基于电子表格做出的数字列表, 或者柱状图、饼状图这类简单的图形化展示方式, 很难展现深层次的细节或数据关联关系, 一些重要的特性或者趋势仍埋藏在数字中。要想更加深入地洞察数据、更加直观地展示数据内在联系, 就需要更加先进、更富有展现力的可视化与交互技术。本节介绍以下 7 种数据可视化技术: 2D 展示技术、3D 渲染技术、体感互动技术、虚拟现实技术、增强现实技术、可穿戴技术和可植入设备。

13.2.1 2D 展示技术

2D 展示技术, 包括标准图表 (柱状图、折线图、饼状图等)、时间序列 (Times Series)、层级树状图 (Hierarchical Tree Map)、时间轴、地图、网络图、信息图等。近几年涌现出了一大批基于 2D 展示技术的数据可视化服务公司。以 Google 为代表的几家公司提供的可视化服务尤其突出。

谷歌的 Charts 提供了用户在网页上以图形方式展示数据的接口, 既支持简单的线图, 也支持复杂的层级树状图等, 采用 JavaScript 就能嵌入网页中。这项服务用起来相当简单, 不用安装任何软件, 只要使用浏览器即可。

要实现复杂数据的展示, 可以在 HTML 文档中使用 JavaScript 语句设置对应的参数, 使用十分方便。Charts 支持饼状图、折线图、柱状图、区域填充图、散点图、维恩图、仪表盘等多种形式, 如图 13.19 所示, 并可以设置图中各部分的颜色、形状、间隔等细节。

D3 (Data Driven Documents) 是一个以操作基于数据的文档资料为主的 JavaScript 库, 支持 CSS3、HTML5 以及 SVG 多种形式的渲染。D3 能够帮助我们快速地把数据转化为图形, D3 强调 Web 标准, 结合强大的可视化组件和基于 DOM 操作的数据驱动方法可以在所

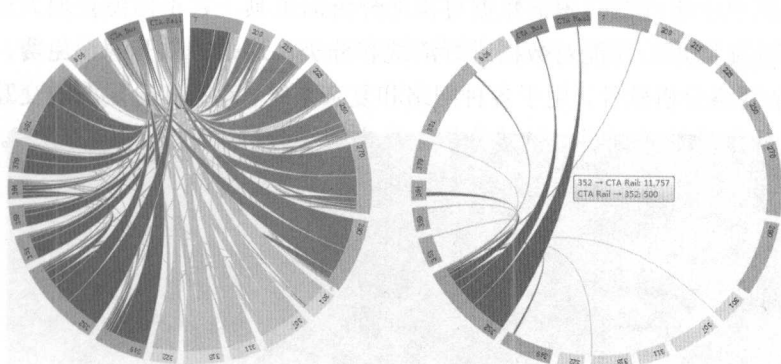


图 13.21 Visualization.org 示例：芝加哥公共交通客流量图

Visual.ly 是信息图领域的另一个重要贡献者。Visual.ly 的主要定位是成为“信息图设计师的在线集市”，它也提供了大量信息图模板。Visual.ly 允许用户提取公共数据（比如 Twitter 话题标记或 Facebook 信息流），然后选择模板，就可以立即生成可视化图标信息。自 2011 年发布以来，已有数百万用户使用 Visual.ly 创建基于 Twitter 的信息图表。

在信息可视化领域，CartoDB 是一个侧重于研究基于地图的数据可视化技术的网站。CartoDB 提供最简单的网络数据导入方式，可以轻松地把表格数据和地图关联起来，实现位置数据可视化；简单易用的设计工具可帮助用户在 Web 上实现漂亮优雅的数据可视化，并可以和团队分享可视化结果或者发布到网络上；可以直接将地图和地理空间分析结果集成到用户的网站。例如，输入 CSV 通信地址文件，CartoDB 可以将地址字符串自动转换为经纬度数据并在地图上标记出来。其创始人 Javier Dela Tore 受到英国《卫报》网站上一幅地图的启发，采用国际非营利性科学组织气象学会的数据用 CartoDB 软件制作了一张记录了从公元前 2300 年开始的陨石坠落地球的热度图，如图 13.22 所示。这张图可以让人们直观地看到那些曾经坠落在地球上的陨石分布情况。

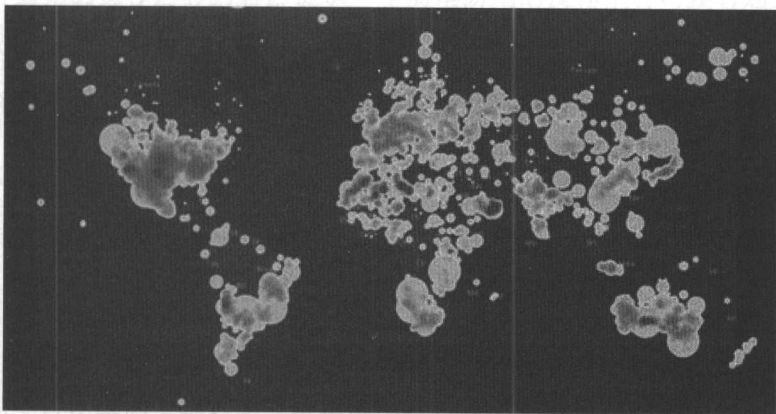


图 13.22 CartoDB 的全球陨石坠落热度图

Gephi 是侧重于进行社交图谱数据可视化分析的工具，它不但能处理大规模数据集并生成漂亮的可视化图形，还能对数据进行清洗和分类。这是一款开源、免费、跨平台的基于 JVM 的复杂网络分析软件，用于各种网络和复杂系统、动态和分层图的交互可视化与探测。它可用于探索性数据分析、链接分析、社交网络分析、生物网络分析等。如图 13.23 所示就是用 Gephi 制作的一个社交网络关系图。

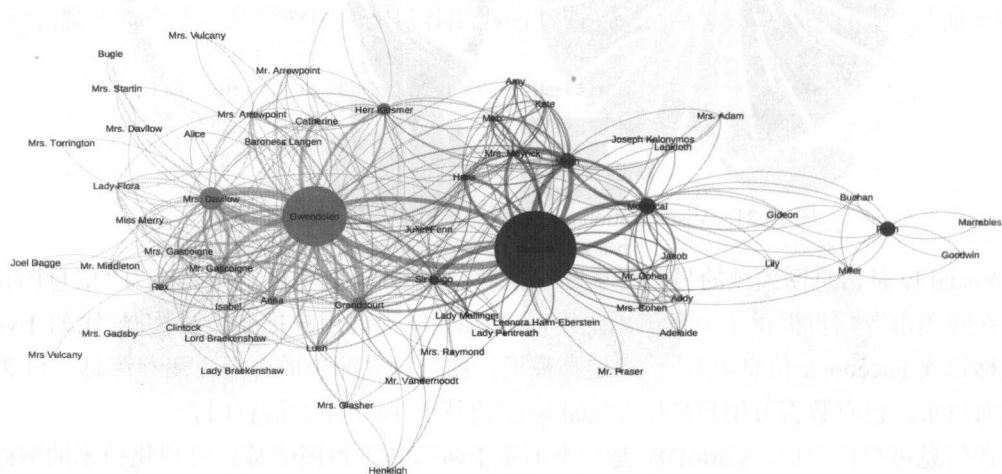


图 13.23 用 Gephi 制作的社交网络关系图

可视化在数据分析领域虽然不是最核心的技术，但是它对人们直观地理解和洞察数据具有十分重要的作用，尤其是在大数据时代，面对纷繁杂乱的非结构化数据、数千万甚至数亿的记录，如果没有可视化工具，想直接从数据中分析查找规律将会是一件非常困难甚至痛苦的事情。随着 Hadoop 等分布式计算平台在大数据计算方面的迅速发展，基于 Hadoop 架构的可视化应用平台也得到了快速发展，为数据科学家和普通商务用户提供了简单易用的工具来处理和展示大数据信息。这些工具中比较突出的有 Ayasdi、Datameer、Tresata、Platfora、ZoomData 等。在这些厂商的推动下，“大数据可视化服务”的发展也如火如荼。基于云技术的大数据应用及可视化服务，使得缺乏大数据专业技术的中小型企业有机会使用大数据分析处理技术及可视化技术，而无须花费巨资去购买相关的硬件和软件，可以节约大量资金，具有很好的市场发展前景。

Ayasdi 来自印第安语，是寻找的意思。来自斯坦福大学的 3 位联合创始人 Gurjeet Singh、Gunnar Carlsson 和 Harlan Sexton 一直致力于将拓扑学的研究方法应用于数据分析。2008 年，他们联合成立了 Ayasdi 公司。Ayasdi 成立以后，就获得了 DARPA（美国国防部高级研究项目组）350 万美元的资助。随后，Ayasdi 的综合了机器学习和拓扑数据分析的技术引起了硅谷投资界的关注。Ayasdi 的底层使用的是 HBase 数据存储，然后利用拓扑数据分析技术和上百种机器学习的算法来处理复杂的数据集，最终确定数据节点之间的相似度。Ayasdi 的技术有一个重要的特点，它不像别的系统需要类似搜索查询式的语句，Ayasdi 可以自动从数据中发现隐藏的模式。Ayasdi 的一个应用方向是医学研究领域，Mount Sinai 医

学院基因与多尺度生物学系的主任 Eric Schadt 带领一个团队,利用 Ayasdi 的技术进行了一些疾病的遗传倾向研究,而且利用 Ayasdi 的数据分析技术,帮助发现了乳腺癌的 14 个变种,如图 13.24 所示。

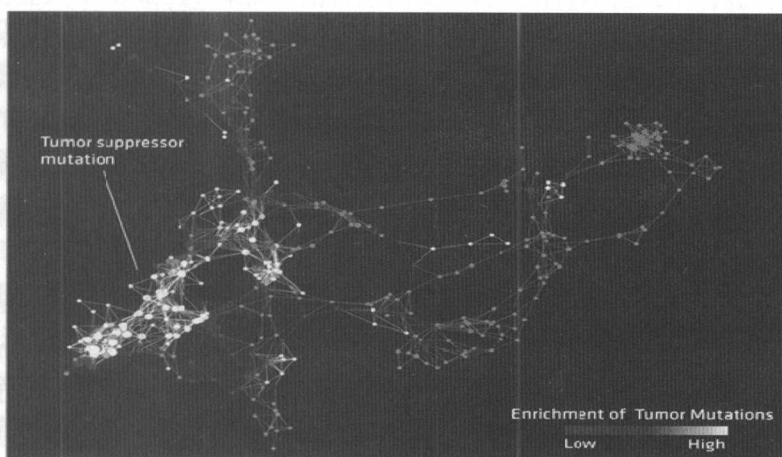


图 13.24 Ayasdi 肿瘤研究

Datameer 采用了大家比较熟知的类似于电子表格的界面,结合其基于 Hadoop 的商务智能平台,允许用户使用或分析存储在 Hadoop 上的数据。

Tresata 的云平台采用 Hadoop 来处理和分析其客户的大量财务数据,并且借用第三方的数据,如股票市场数据来丰富数据内容;另外,它还通过按需虚拟化为银行、金融数据公司以及其他的金融服务单位反馈分析结果。

Platfora 是一个提供原始数据准备、内存加速和丰富可视化功能的基于 Hadoop 的端到端软件平台,目的是“把 Hadoop 平民化”。该公司认为只有对冗杂的数据进行有效处理、视觉化,让数据编程普通用户都能看懂,大数据才能真正具备商业价值。Platfora 在 Hadoop 的基础上进行数据的操作,并为客户提供一个简单易用的操作平台,使普通用户不需要专业开发人员就可以利用 Hadoop 平台的强大计算功能进行全样本的大数据分析和可视化,发现更多数据中的价值。因为 Hadoop 有很多不同的发行版本,所以 Platfora 的重点之一就是确保它能够在所有的发行版上运行,这样就大大降低了 Hadoop 的使用门槛,让更多的人能够体验 Hadoop 的技术优势,实现真正意义上的平民化。目前该公司已经获得了 2000 万美元的 B 轮融资。

ZoomData 是为数不多的支持移动设备的数据分析公司,它的数据可视化系统能够将实时的大数据流转化为触屏友好、艺术感十足的三维数据。平板电脑用户可以用手指缩放数据可视化界面,随着缩放比例不同,数据将实时进行更新。ZoomData 的数据可视化技术支持多种数据源,包括社交媒体、企业应用系统及 Hadoop HDFS 数据。

13.2.2 3D 渲染技术

3D 渲染技术是近年来发展迅速和备受关注的行业,在数字娱乐、虚拟现实、工业设计、实时仿真、数字城市等各个领域都有着十分广泛的应用。在数字娱乐领域,提到 3D 动画渲染,人们马上就会联想到皮克斯公司。皮克斯是一家专门制作计算机动画的公司,其所制作的《怪兽公司》、《虫虫危机》、《海底总动员》、《料理鼠王》等动画电影系列,都受到全球观众热捧。皮克斯出名的另一个重要原因是他的老板是苹果公司前总裁斯蒂夫·乔布斯。1986 年,乔布斯以 1000 万美元的价格收购了有名的乔治·卢卡斯的计算机动画部,成立了皮克斯动画工作室。皮克斯也是世界上第一部全计算机制作的动画电影《玩具总动员》的制作公司。该片 1995 年在全美上映,以 1.92 亿美元的票房刷新了动画电影的纪录,成为当年美国票房冠军,也缔造了全球 3.6 亿美元票房的纪录,还为导演约翰·拉塞特赢得了奥斯卡特殊成就奖。该片的巨大成功还促成了皮克斯的老板斯蒂夫·乔布斯被濒临倒闭的苹果公司又请了回去,这才有了在电子科技领域影响了世界的 iPad 和 iPhone。

3D 特技在科幻影片《阿凡达》中更是被发挥得淋漓尽致。该片也因为震撼的特技效果而获得了全球电影史上的最高票房,并获得了第 82 届奥斯卡最佳艺术指导、最佳摄影和最佳特效 3 项奖项,以及第 67 届金球奖最佳导演奖和最佳影片奖。

在工业设计领域,目前在建筑、飞机、轮船、汽车、机床等设备的设计中已经普遍用到 3D 技术,它使设计师可以在屏幕上随时变更设计方案,进行快速验证。在当今的数字城市、智慧城市建设中,3D 技术也展现了巨大的能量,它不仅能够模拟整个城市、园区、建筑、室内的建设效果,还能结合控制参数,实时仿真出动态反应场景,比如变电站的控制、水库的监测等。采用 3D 技术,可以大大节省实际生产和制造的时间和成本,同时直观地展示出最终的效果,能够高效地进行互动调整等。图 13.25 就是 3D 技术渲染的电厂效果图。

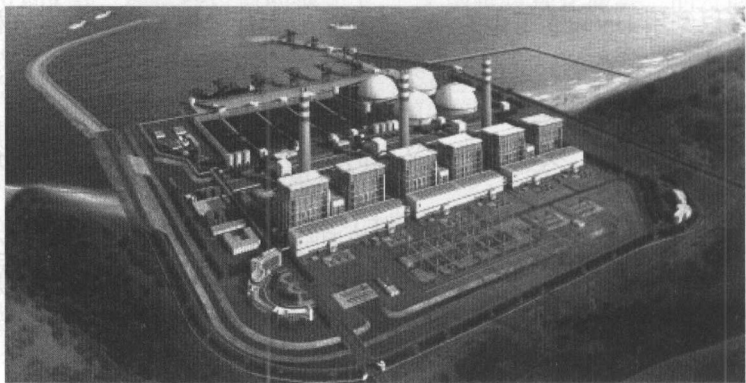


图 13.25 3D 技术渲染的电厂效果图

比较常用的 3D 制作和设计软件有 Adobe Flash、Maya、Autodesk 3ds Max、SketchUp 等。

1. Adobe Flash

Adobe Flash 是一款集动画创作与应用程序开发于一身的软件,新版本 Adobe Flash 为数字动画、交互式 Web 站点、桌面应用程序以及手机应用程序提供了功能全面的创作和编辑环境。Adobe Flash 广泛用于创建吸引人的应用程序,它们包含丰富的视频、声音、图形和动画。设计师可以在 Adobe Flash 中创建原始内容或者从其他 Adobe 应用程序(如 Photoshop 或 Illustrator)导入素材,快速设计简单的动画,以及使用 Adobe ActionScript 3.0 开发高级的交互式项目。Flash 3D 具有可在线浏览 3D 模型、跨平台、更自由的浏览模式(可通过鼠标及鼠标滚轮、键盘放大/缩小浏览、全屏浏览,并且具有效果不变的优点)。Adobe Flash 可用的 3D 引擎有很多,常见的有 Away 3D、Alternativa 3D、Flare 3D、CopperCube、Unity 3D、Papervision 3D 等。

(1) Away 3D

Away 3D 具有一个可视化编辑场景及模型的工具——Prefab3D,这个运用 Adobe AIR 开发的工具功能相当强大,开发者和设计人员可以方便地对三维场景进行材质贴图、编辑光照及设置动画等,并输出为 Away 3D 使用的文档。

引擎相关特性:

- ① 支持加载大多数流行 3D 文件,如 Collada、OBJ 等;
- ② 拥有可视化编辑场景及模型的免费工具 Prefab3D;
- ③ 具有功能全面的资源加载、事件处理、光照、摄像机、骨骼动画及音效处理等。

(2) Alternativa 3D

在 Molehill 出来之前,用该引擎开发的 Tanki Online 就让大家惊艳。Adobe Max 大会上的 3D 赛车就出自 Alternativa 3D 引擎。

引擎相关特性:

- ① 支持加载大多数流行 3D 文件,如 Collada、OBJ 等;
- ② 拥有 3ds Max 2010 输出插件;
- ③ 可以类似 DisplayObject 方式方便地管理 3D 对象;
- ④ 可进行高效的三维深度排序;
- ⑤ 具有光照系统、鼠标交互、多摄像机系统等。

(3) Flare 3D

Flare 3D 是一个创建 Flash 3D 游戏的引擎。其最大特色是具有较完整的 Flash 3D 游戏开发工作流程。

引擎相关特性:

- ① 支持导入 3ds Max 模型;
- ② 能可视化地对场景及模型进行编辑、贴图;
- ③ 具有光照系统、骨骼、摄像机系统等;
- ④ 具有比较直观的开发流程。

(4) CopperCube

CopperCube 是一个具有 3D 引擎及编辑器的开发工具，开发者可以通过它将游戏及程序发布为多种格式。

引擎相关特性：

- ① 能发布为多种格式；
- ② 能支持多达 20 多种的三维模型格式；
- ③ 能可视化地对场景及模型进行编辑、贴图、动作设置等；
- ④ 代码编写量小，号称无须编程即可创建 3D 应用；
- ⑤ 有比较直观的开发流程。

(5) Unity 3D

Unity 3D 是由 Unity Technologies 开发的一个让用户轻松创建诸如三维视频游戏、建筑可视化、实时三维动画等类型互动内容的多平台的综合型游戏开发工具，是一个全面整合的专业游戏引擎。

引擎相关特性：

- ① Unity 对 DirectX 和 OpenGL 拥有高度优化的图形渲染管道；
- ② Unity 的着色器系统整合了易用性、灵活性和高性能；
- ③ 低端硬件也可流畅运行广阔茂盛的植被景观；
- ④ 实时三维图形混合音频流、视频流；
- ⑤ 光影 Unity 提供了具有柔和阴影与烘焙 Lightmaps 的高度完善的光影渲染系统。

(6) Papervision 3D

Papervision 3D 是较早的 3D 引擎，性能不错，但是相对来说，模型却不多，而且 Camera 也不是很好用，没有默认的控制器的。

引擎相关特性：

- ① 支持 ASE 和 DAE 格式的 3D 模型；
- ② 支持众多材质方式。

各种引擎开源和收费情况比较见表 13.1。

表 13.1 各种引擎开源和收费情况比较

	Away 3D	Alternativa 3D	Flare 3D	CopperCube	Unity 3D	Papervision 3D
使用条件	免费	免费	商业	商业	商业	商业
授权方式	开源	开源	非开源	非开源	非开源	开源

2. Maya

Maya 是世界顶级的三维动画软件，被广泛用于电影、电视、广告、电脑游戏和电视游戏等的数位特效创作，曾获奥斯卡科学技术贡献奖等殊荣。Maya 功能完善，易学易用，制作效率极高，渲染真实感很强，是电影级别的高端制造软件。掌握 Maya 后，会极大地提高制作效率和产品品质，得到仿真的角色动画，渲染出电影一般的真实效果。很多 3D 电

影如《海底总动员》、《最终幻想》、《指环王》、《黑客帝国》等的特效都出自 Maya。2005 年 10 月 4 日, 生产 3D Studio Max 的 Autodesk 软件公司宣布正式以 1.82 亿美元收购生产 Maya 的 Alias, 所以 Maya 现在是 Autodesk 的软件产品。当前 Maya 集成了 Alias、Wavefront 最先进的动画及数字效果技术。它不仅具有一般三维和视觉效果制作功能, 而且还与最先进的建模、数字化布料模拟、毛发渲染、运动匹配技术相结合。Maya 可在 Windows NT 与 SGI IRIX 操作系统上运行, 它的主要应用领域包括四个方面一是平面图形可视化, 它可增强平面设计产品的视觉效果, 强大的功能开阔了平面设计师的应用视野; 二是网站资源开发; 三是电影特技; 四是游戏设计及开发。

3. Autodesk 3ds Max

3D Studio Max, 常简称 3ds Max 或 Max, 是 Autodesk 公司开发的基于 PC 系统的三维动画渲染和制作软件。其前身是基于 DOS 操作系统的 3D Studio 系列软件。在 Windows NT 出现以前, 工业级的计算机图形学 (Computer Graphics, CG) 制作被 SGI 图形工作站所垄断, 但是 3D Studio Max + Windows NT 组合的出现一下子降低了 CG 制作的门槛, 首先运用于电脑游戏中的动画制作, 之后更进一步开始参与影视片的特效制作, 例如《X 战警 II》、《最后的武士》等。当前其广泛应用于广告、影视、工业设计、建筑设计、三维动画、多媒体制作、游戏、辅助教学以及工程可视化等领域。3ds Max 凭借 PC 系统的低配置要求、安装插件可增强或扩展功能、强大的角色动画制作能力、可堆叠的建模步骤、上手容易等特点得到了迅速的普及推广, 是当前国内应用非常广泛的软件。

4. 谷歌 SketchUp

SketchUp 是一个极受欢迎并且易于使用的 3D 设计软件, 官方网站将它比喻为电子设计中的“铅笔”。它的主要优势就是使用简便, 人人都可以快速上手, 并且用户可以将使用 SketchUp 创建的 3D 模型直接输出至 Google Earth 里。

Google SketchUp 具有丰富的模型资源, 在设计中可以直接调用、插入、复制。同时 SketchUp 集成了一套精简而强健的工具集和一套智慧导引系统, 大大简化了 3D 绘图的过程, 让使用者专注于设计上。现在 SketchUp 及其组件资源已经广泛应用于室内、室外、建筑等领域中。

SketchUp 的主要特点:

- ① 独特简洁的界面, 可以让设计师快速掌握;
- ② 适用范围广阔, 可以应用在建筑、规划、园林、景观、室内以及工业设计等领域;
- ③ 具有方便的推拉功能, 设计师通过一个图形就可以方便地生成 3D 几何体, 无须进行复杂的三维建模;
- ④ 快速生成任何位置的剖面, 使设计者清楚了解建筑的内部结构, 可以随意生成二维剖面图并快速导入 AutoCAD 进行处理;
- ⑤ 与 AutoCAD、Revit、3ds Max、PIRANESI 等软件结合使用, 快速导入和导出 DWG、DXF、JPG、3DS 格式文件, 实现方案构思, 使效果图与施工图绘制完美结合, 同时提供

AutoCAD 和 AachiCAD 等设计工具的插件;

- ⑥ 自带大量门、窗、柱、家具等组件库和建筑肌理边线需要的材质库;
- ⑦ 轻松制作方案演示视频动画, 全方位表达设计师的创作思路;
- ⑧ 具有草稿、线稿、透视、渲染等不同显示模式;
- ⑨ 准确定位阴影和日照, 设计师可以根据建筑物所在地区和时间实时进行阴影和日照分析;
- ⑩ 可简便地进行空间尺寸和文字的标注, 并且标注部分始终面向设计者。

13.2.3 体感互动技术

体感互动技术是通过硬件互动设备、体感互动系统软件及三维数字内容来感应站在设备前的操作者, 当操作者做出一定动作时, 设备所显示的画面也相应发生变化。

如图 13.26 所示, 玩家手持游戏手柄进行“网球体感游戏”, 玩家的手部击球动作可完全用来模拟并控制游戏里游戏角色的球路, 那么玩家手部的动作与游戏角色球路的对应是如何实现的呢? 原理在于玩家手上的手柄能获取玩家手部的各种物理参数, 如加速度、角速度、位移等, 然后进一步通过算法, 将这些物理参数转化为人体在空间中的三个平移量以及三个旋转量, 如此一来便可将手部在空间中的各种动作(平移+旋转)完全描述出来, 接着再将此平移及旋转量传输给游戏角色, 游戏角色便可与玩家做出相同的动作。因此所谓的“体感游戏”, 便是通过各种传感器捕捉人体的肢体动作(平移+旋转), 并将所计算出的肢体动作对应于游戏角色的反应, 使玩家的动作与游戏角色的反应呈现 1:1 拟真的对应。

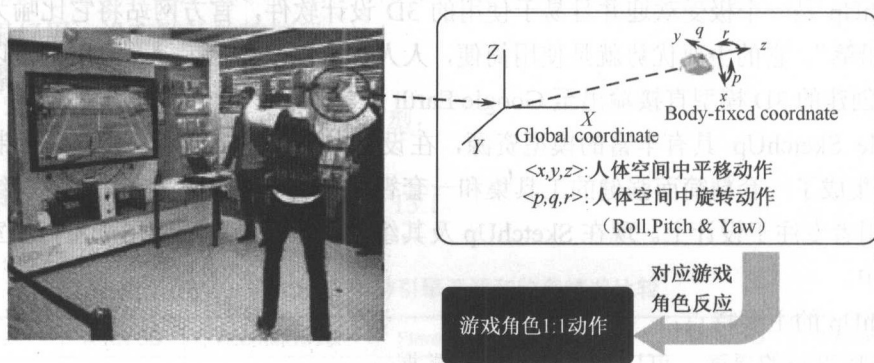


图 13.26 体感互动游戏原理示意图

体感互动技术的优势在于人们可以很直接地使用肢体动作, 与周边的装置或环境互动, 而无须使用任何复杂的控制设备。体感互动技术被广泛应用在数字娱乐、媒体广告、医疗、教育培训、工业设计及控制等各个领域。

按照体感方式和原理的不同, 体感技术主要可分为三大类: 惯性感测、光学感测、惯性及光学联合感测。

1. 惯性感测

惯性感测是以惯性传感器为主要感测设备, 利用重力传感器、陀螺仪以及磁传感器等来感测使用者肢体动作的物理参数(加速度、角速度以及磁场等), 再根据这些物理参数求得使用者在空间中的各种动作。主要代表厂商为 Logitech, 其在 2007 年推出了空间鼠标(MxAir), 使用三轴重力传感器以及两轴陀螺仪, 可感测使用者在空间的手部动作, 并将此动作转化为鼠标在屏幕上垂直方向与水平方向的位移。

2009 年, 苹果智能型手机拉开了手机体感游戏热门下载的序幕, 许多使用惯性传感器来适配的体感游戏不断地出现。其中, iPhone 使用了以三轴重力感测以及三轴磁传感器为主的惯性感测。2010 年, 基于未来即将陆续推出拥有重力传感器、磁传感器和陀螺仪的智能型手机, CyWee 发展了面向这三种传感器的特有算法, 称为九轴混合感测算法(9-axis Sensor Fusion Technology)。所谓的九轴, 指的便是可量测空间中三轴向的重力传感器、可量测三轴向的磁传感器, 以及可量测三轴向的陀螺仪。此算法可克服传统上仅使用单一传感器的缺点, 进而达成更精确的空间动作捕捉体感体验。

2. 光学感测

2005 年, Sony 推出了光学感应套件——EyeToy, 主要是通过光学传感器获取人体影像, 再将此人体影像的肢体动作与游戏中的内容互动。2010 年, Microsoft 发布了跨时代的全新体感感应套件——Kinect, 号称无须使用任何体感手柄, 便可达到体感的效果, 而比起 EyeToy 更为进步的是, Kinect 同时使用激光及摄像头(RGB)来获取人体影像信息, 可捕捉人体 3D 全身影像, 具有比 EyeToy 更为进步的深度信息, 而且不受任何灯光环境限制。2013 年, Leap 发布了 Leap Motion, 这是一种精度更高的手动控制技术, 能通过手指直接控制计算机, 包括图片缩放、移动、旋转、精准控制、指令操作、隔空书写等。通过放在键盘和显示器之间的金属盒, 就能让任何一个用户通过简单的手势完成人机交互。Leap Motion 的主要原理是使用红外 LED 和两个灰度摄像头采集手掌数据, 并生成 3D 数据。系统 150° 超宽幅的空间视场, 可追踪全部 10 只手指, 精度高达 1/100mm, 每秒 200 帧的追踪速度可以实现有效空间范围内的任何细微动作捕捉(图 13.27)。

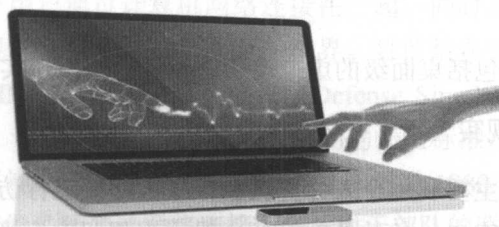


图 13.27 Leap Motion

3. 惯性及光学联合感测

主要代表厂商为 Nintendo 及 Sony。2006 年 Nintendo 所推出的 Wii, 主要是在手柄上

放置一个重力传感器（用来侦测手部三轴向的加速度），以及一个红外线传感器（用来感应电视屏幕前方的红外线发射器信号），主要可用来侦测手部在垂直及水平方向的位移，从而操控一个空间鼠标。这样的配置往往只能侦测一些较为简单的动作，因此 Nintendo 在 2009 年推出了 Wii 手柄的加强版——Wii Motion Plus，其在原有的 Wii 手柄上再插入一个三轴陀螺仪，如此一来便可更精确地侦测人体手腕旋转等动作，强化了在体感方面的体验。至于在 2005 年推出 EyeToy 的 Sony，也不甘示弱地在 2010 年推出了游戏手柄 Move，主要配置包含一个手柄及一个摄像头，手柄包含重力传感器、陀螺仪以及磁传感器，摄像头用于捕捉人体影像，结合这两种传感器，便可侦测人体手部在空间中的移动及转动。

13.2.4 虚拟现实技术

虚拟现实（Virtual Reality，VR）技术是由美国 VPL 公司创始人拉尼尔（Jaron Lanier）在 20 世纪 80 年代初提出的，也称灵境技术或人工环境。作为一项尖端科技，虚拟现实集成了计算机图形技术、计算机仿真技术、人工智能、传感技术、显示技术、网络并行处理等技术的最新发展成果，是一种由计算机生成的高技术模拟系统，它最早源于美国军方的作战模拟系统，20 世纪 90 年代初逐渐为各界所关注并且在商业领域得到了进一步的发展。这种技术的特点在于计算机产生一种人为虚拟的环境，这种虚拟的环境是通过计算机图形构成的三维数字模型，并编制到计算机中去生成一个以视觉感受为主，同时包括听觉、触觉的综合可感知的人工环境，从而使人产生一种沉浸于这个环境的感觉，人可以直接观察、操作、触摸、检测周围环境及事物的内在变化，并能与之发生“交互”作用，使人和计算机很好地“融为一体”，给人一种“身临其境”的感觉。

一般的虚拟现实系统主要由专业图形处理计算机、应用软件系统、输入设备和演示设备等组成。虚拟现实技术的特征之一就是人机之间的交互性（interaction）。为了实现人机之间充分交换信息，必须设计特殊输入工具和演示设备，以识别人的各种输入命令，且提供相应反馈信息，实现真正的仿真效果。不同的项目可以根据实际应用选择使用不同工具，主要包括：头盔式显示器、跟踪器、传感手套、屏幕式或房式立体显示系统、三维立体声音生成装置。

虚拟现实技术主要包括桌面级的虚拟现实、投入的虚拟现实、分布式虚拟现实等。

1. 桌面级的虚拟现实

桌面级的虚拟现实主要利用个人计算机和低级工作站进行仿真，计算机屏幕被用做用户观察虚拟境界的一个窗口，各种外部设备一般用来驾驭虚拟境界，并且有助于操纵在虚拟情景中的各种物体。这些外部设备包括鼠标、追踪球、力矩球等。它要求参与者使用位置跟踪器和另一个手控输入设备，如鼠标、追踪球等，坐在监视器前，通过计算机屏幕观察 360° 范围内的虚拟境界，并操纵其中的物体，但这时参与者并没有完全投入，因为他仍然会受到周围现实环境的干扰。桌面级的虚拟现实最大特点是缺乏完全投入的功能，但

是成本相对低一些，因而应用面比较广。常见桌面虚拟现实技术有：基于静态图像的虚拟现实技术、VRML（虚拟现实造型语言）、桌面 CAD 系统。

2. 投入的虚拟现实

这种方式提供完全投入的功能，使用户有一种置身于虚拟境界之中的感觉。它利用头盔式显示器或其他设备，把参与者的视觉、听觉和其他感觉封闭起来，并提供一个新的、虚拟的感觉空间，利用位置跟踪器、数据手套、其他手控输入设备、声音等使参与者产生一种身在虚拟环境中，并能全心投入和沉浸其中的感觉。常见的沉浸式系统是基于头盔式显示器的系统，这也是目前沉浸度最高的一种虚拟现实系统。在这种系统中，参与虚拟体验者要戴上一个头盔式显示器，视觉、听觉与外界隔绝，根据应用的不同，系统将提供能随头部转动而产生的立体视觉、三维空间。通过语音识别、数据手套、数据服装等先进的接口设备，使参与者以自然的方式与虚拟世界进行交互，如同现实世界一样。2014 年 3 月 Facebook 以 20 亿美元收购 Oculus 后，虚拟现实游戏眼罩 Oculus Rift 成为虚拟现实技术领域的一个热点。这款产品内置 3D 立体显示器和陀螺仪、加速计等惯性传感器，可以实时感知使用者头部的位置，并对应调整显示画面的视角（图 13.28）。

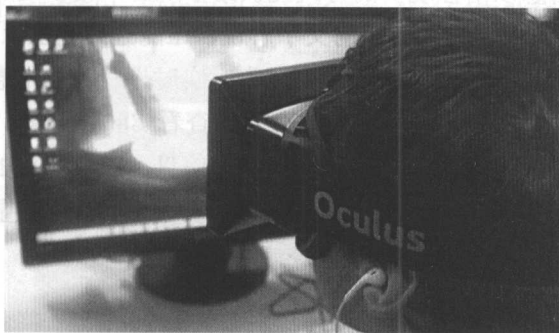


图 13.28 Oculus 产品应用于游戏中

3. 分布式虚拟现实

这种技术可实现多个用户通过计算机网络连接在一起，同时参加一个虚拟空间，共同体验虚拟经历，使虚拟现实提升到了一个更高的境界。目前最典型的分布式虚拟现实系统是作战仿真互联网和 SIMNET。作战仿真互联网（Defense Simulation Internet, DSI）是目前最大的 VR 项目之一。该项目是由美国国防部推动的一项标准，目的是使各种不同的仿真器可以在巨型网络上互连，它是美国国防高级研究计划局 1980 年提出的 SIMNET 计划的产物。SIMNET 由坦克仿真器通过网络连接而成，用于部队的联合训练。通过 SIMNET，位于德国的仿真器可以和位于美国的仿真器运行在同一个虚拟世界中，参与同一场作战演习。

13.2.5 增强现实技术

增强现实 (Augmented Reality, AR) 是在虚拟现实的基础上发展起来的, 它是把计算机产生的虚拟的物体和场景叠加到现实场景中, 用这种混合的模式增强用户对场景的感知。它是一种全新的人机交互技术, 利用这样一种技术, 可以模拟真实的现场景观, 它是交互性和构想为基本特征的计算机高级人机界面。使用者不仅能够通过虚拟现实系统感受到在客观物理世界中所经历的“身临其境”的逼真性, 而且能够突破空间、时间以及其他客观限制, 感受到在真实世界中无法亲身经历的体验。

AR 技术包含了图形图像学、可视化技术、实时交互技术、多传感器融合等新技术和新手段, 系统具有三个突出的特点: 真实世界和虚拟世界的信息集成, 实时交互性, 在三维尺度空间中增添定位虚拟物体。AR 技术可以广泛应用于军事、医疗、古迹保护、建筑、教育、工程、影视、娱乐等领域。在军事方面, 用于尖端武器、飞行器的研发、虚拟训练等; 在医疗领域, 用于帮助医生进行手术部位的精确定位; 在古迹复原和数字化文化遗产保护领域, 将文化古迹的信息以增强现实的方式提供给参观者, 使参观者不仅可以通过专用的显示器 (头盔式、眼镜式等) 看到古迹的文字解说, 还能看到遗址上残缺部分的虚拟重构; 在电视领域, 通过 AR 技术可以在转播体育比赛时实时地将辅助信息叠加到画面中; 在商品展示中, 可以将家具和电器叠加到顾客的客厅中查看实时效果等。

增强现实系统设计最基本的问题就是实现虚拟信息和现实世界的融合, 显示技术是系统的基本技术之一。增强现实的显示技术主要分为以下几类: 头盔显示器显示、投影式显示、手持式显示器显示和普通显示器显示。

1. 头盔显示器显示 (Head-mounted Display, HMD)

头盔显示器采用影像叠加的方式实现, 所以也叫透视式 (see-through) 头盔显示器。利用它能够看到周围的真实环境, 沉浸感更强, 是应用较广的一种显示方式。

透视式头盔显示器一般分为视频透视式 (Vedio see-through) 和光学透视式 (Optical see-through)。前者利用摄像机对真实世界进行同步拍摄, 将信号送入虚拟现实工作站, 在虚拟工作站中将虚拟场景生成器生成的虚拟物体同真实世界中采集的信息融合, 然后输出到头盔显示器。而后者则利用投影装置将虚拟物体投射到透明眼镜上, 这些虚拟物体与用户通过眼镜看到的真实景象进行融合, 实现增强效果。

还有一种更为奇特的方法是虚拟视网膜显示 (Virtual Retinal Display, VRD) 技术。华盛顿大学的人机界面实验室 (HIT Lab) 研究出的 VRD 通过将低功率的激光直接投射到人眼的视网膜上, 将虚拟物体添加到现实世界中来。

2. 投影式显示 (Projection Display)

投影式显示是将虚拟的信息直接投影到要增强的物体上, 从而实现增强。日本 Chuo

大学研究出的 PARTNER 增强现实系统可以用于人员训练，并且使一个没有受过训练的试验人员通过系统的提示，成功地拆卸了一台便携式 OHP（Over Head Projector）。

另外一种投影式显示方式是采用放在头上的投影机（Head-mounted Projective Display, HMPD）来进行投影。美国伊利诺伊州立大学和密歇根州立大学的一些研究人员研究出一种 HMPD 的原型系统。该系统由一个微型投影镜头、一个戴在头上的显示器和一个双面自反射屏幕组成。由计算机生成的虚拟物体显示在 HMPD 的微型显示器上，虚拟物体通过投影镜头折射后再由与视线成 45° 角的分光器反射到自反射的屏幕上面。自反射的屏幕将入射光线沿入射角反射回去进入人眼中，从而实现了虚拟物体与真实环境的重叠。

3. 手持式显示器显示（Hand Held Display, HHD）

通过采用摄像机等其他辅助部件，一些增强现实系统采用了手持式显示器。美国华盛顿大学的人机界面实验室设计出了一个便携式的 MagicBook 增强现实系统。该系统采用一种基于视觉的跟踪方法把虚拟的模型重叠在真实的书籍上，产生一个增强现实的场景。同时该系统也支持多用户的协同工作。日本的 Sony 计算机科学实验室也研究出了一种手持式显示器，利用这种显示器，构建了 TransVision 协同式工作环境。宜家公司推出了 3D 增强现实技术的应用，通过这款应用，用户可以看到家具在自己家中的模拟 3D 效果（图 13.29）。



图 13.29 增强现实技术示例

4. 普通显示器显示（Monitor-based Display）

在基于普通显示器的方案中，摄像机摄取的真实世界图像输入计算机中，与计算机图形系统产生的虚拟景象合成，并输出到显示器，用户可以从屏幕上看到最终的增强场景图片。这种 AR 技术实现相对简单，比如通过 AR 技术试戴眼镜，用户不用摘掉自己的眼镜，转动头部就可以看到叠加在自己眼眶上的各种不同眼镜的效果。

13.2.6 可穿戴技术

可穿戴技术主要是探索和创造能直接穿在身上，或者整合进用户的衣服或配饰的设备的科学技术。可穿戴技术是 20 世纪 60 年代美国麻省理工学院媒体实验室提出的创新技术，利用该技术可以把多媒体、传感器和无线通信等技术嵌入人们的衣着中，可支持手势和眼

动操作等多种交互方式。其目的是通过“内在连通性”实现快速的数据获取，通过超快的分享内容能力高效地保持社交联系，摆脱传统的手持设备而获得无缝的网络访问体验。

20 世纪 60 年代，可穿戴技术逐渐兴起；到了 70 年代，发明家 Alan Lewis 打造的配有数码相机功能的可穿戴式计算机能预测赌场轮盘的结果。1977 年，Smith-Kettlewell 研究所视觉科学院的 C.C.Colin 为盲人做了一款背心，把头戴式摄像头获得的图像通过背心上的网格转换成触觉意象，让盲人也能“看”得见。自 2012 年 4 月谷歌公司宣布其 Google Project Glass 的未来眼镜研发项目后，各大科技公司纷纷在可穿戴技术应用上加大研发力度。2013 年，苹果密集曝光了其智能手表 Apple Watch 的一系列新功能并于 2014 年进行了产品发布；索尼于 2013 年 8 月底推出了 Smart Watch 的第二代产品；三星则在 2013 年 9 月推出了智能手表产品 Galaxy Gear。

可穿戴技术是近几年科技节最热门的趋势之一。在每一次大型科技盛会上，面向个人消费者的可穿戴设备数量都呈现指数级增长的趋势，从智能手环、智能手表、智能手套到智能眼镜、智能头盔等。可穿戴设备不仅仅是硬件设备，更是可以通过软件支持及数据交互、云端交互来实现强大功能拓展的设备。可穿戴设备具有快速的信息抓取、处理和查询能力，以及更准确的判断决策能力，在这种设备的帮助下，人们的行为模式和行动效率也将得到改善和提高。

目前已经问世和即将问世的可穿戴设备，基本包括四大类：

- ① 运动和健康辅助设备，如爱普生 Pulsense 系列、Nike+Fuelband、FitbitFlex，以及国内的咕咚手环、大麦计步器、小米手环等；
- ② 可以不依附于智能手机的独立智能设备，如 Apple Watch、果壳智能手表；
- ③ 互联网辅助产品，如 Google Glass、百度 Eye 类产品；
- ④ 与物联网密切相关的体感设备，如 MYO 等。

爱普生发布的 Pulsense 系列可穿戴设备，包括智能手表和智能手环。这些产品整合了爱普生公司行业领先的独创生物感应技术与基于云系统的服务，可满足穿戴消费品市场健身、健康和运动需求。Pulsense 系列中的 PS-500 智能手表和 PS-100 智能手环，是具有生物识别功能的腕部感应可穿戴设备，对心率、活动强度、卡路里燃烧与睡眠模式具有监控与数据存储功能，是理想的日常活动穿戴产品，也是心脏健康记录设备，并可以通过智能手机应用软件，将存储的自身数据传到在线健康和健身服务软件中，或通过电脑上传软件传输这些数据。

2012 年 6 月 28 日，谷歌发布了一款穿戴式 IT 产品——谷歌眼镜。该设备由一块位于右眼侧上方的微缩显示屏、一个在右眼外侧平行放置的 720P 画质摄像头、一个位于太阳穴上方的触摸板，以及喇叭、麦克风、陀螺仪传感器和可以支撑 6 小时电力的内置电池构成，结合了声控、导航、照相与视频聊天等功能。

苹果于 2014 年 9 月发布的智能手表——Apple Watch，也是一款可穿戴的智能设备，分为运动款、普通款和定制款三种，采用蓝宝石屏幕。这款手表内置了 iOS 系统，并且支持 Facetime、WiFi、蓝牙、Airplay、电话、语音回短信、连接汽车、天气、航班信息、地图

导航、播放音乐、测量心跳、计步等几十种功能，是一款全方位的健康和运动追踪设备（图 13.30）。

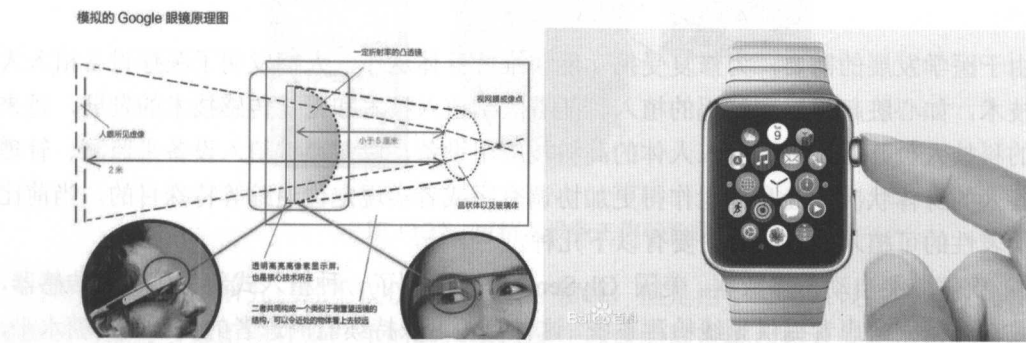


图 13.30 谷歌眼镜原理图与 Apple Watch

BrainLink 意念头箍是由深圳市宏智力科技有限公司专为 iOS 系统研发的配件产品，它是一个安全可靠、佩戴简易方便的头戴式脑电波传感器（图 13.31）。作为一款可佩戴式设备，它可以通过蓝牙无线连接智能手机、平板电脑、手提电脑、台式电脑或智能电视等终端设备，配合相应的应用软件就可以实现意念力互动操控。BrainLink 采用了国外先进的脑机接口技术，其独特的外观设计、强大的培训软件深受广大用户的喜爱。



图 13.31 BrainLink 意念头箍

随着可穿戴技术越来越重要，利用无线连接技术实现设备与智能手机的互连将会成为开发这些设备应用潜力的关键所在。例如，借助近场通信（NFC）技术，消费者可以购买新型可穿戴设备并将其方便地连接到智能手机，进行快速安全的通信，不需要其他复杂的菜单或烦琐的设置过程；借助 Bluetooth Smart 和 WiFi 技术，消费者可以从可穿戴设备中获取数据（例如消耗的卡路里、心率等），并将数据传送到智能手机或云端，而不会消耗太多电量；借助 WiFi 直连技术，消费者可以直接将两个 WiFi 设备连接在一起，不需要接入点或计算机；将可穿戴设备与定位技术结合起来，可以实现一些有趣的新应用功能，比如医生可以在临床环境中跟踪患者的情况，零售商可以向消费者发送有针对性的广告信息等。

13.2.7 可植入设备

由于医学发展的需要，为修复受损功能和维持身体运作，人们发明了医疗设备植入人体的技术，如心脏起搏器和耳蜗的植入。随着医学植入技术和无线传感技术的发展，越来越多的科技人员开始研究可植入人体的高科技应用设备，以便通过植入设备来监测、管理和改善人的身体状况，让身体运作得更加协调有序或者实现定位追踪等特殊目的。当前比较有代表性的可植入设备研究主要有以下几种。

① 疾病监测自动传输数据。美国 GlySens 公司研制了一种植入式新型葡萄糖传感器，可向数百万糖尿病患者提供无线检测系统。这种设备可以持续监测患者的皮下葡萄糖水平，而监测结果可以反映血糖浓度。

② 人脑与计算机直连。布朗大学的 BrainGate 团队，深入研究了如何实现人的大脑与电脑对接。初步研究显示，在人脑中植入婴儿版阿司匹林大小的电极，神经信号可被电脑实时解码，并用于操控外部设备。英特尔预测，至 2020 年脑机界面将投入实际应用。

③ 药片发短信给医生。一家总部位于英国的制药公司 Proteus，展示了一种全新的数字药片，通过胃酸或者其他医学技术来提供电力。药丸内置微型处理器，可将人体内的信息，通过短信发送给医生，以帮助医生了解患者的健康状况以及服药情况等。

④ 智能尘埃击退早期癌细胞。智能尘埃是一种带天线的比沙粒更小的微型电脑阵列，它们可在体内进行自我组合以成为需要的网络，用以处理体内的复杂情况。未来，这类纳米级别的器件或许能够击退早期癌细胞，缓解伤口疼痛，甚至加密存储个人敏感信息。而医生也可在患者体内进行无创手术。

⑤ 治愈芯片帮助减肥。目前，波士顿大学正在测试一种智能仿生胰腺，在可植入式针头上附有微型传感器，传感器能与监测血糖水平的智能手机应用建立直接对话。此外，伦敦的科学家也在开发一种与胶囊类似的电路，用于监测肥胖病人的脂肪水平，生成令他们产生饱腹感的遗传物质，并有望取代手术或其他减肥方法。

⑥ 自我验证、自动追踪。该技术可用于个人的身份认证。美国军方正在为士兵植入 RFID 芯片，该技术可在全球范围内自动追踪部队。未来，该技术或许能在打击犯罪、确保选举安全、追踪失踪人口等方面有所建树。

⑦ 皮下射频识别 (RFID) 芯片。研制者将其做成“小型胶囊”，并成功打入使用者体内。这种小型胶囊长 12mm，直径为 2mm，采用具有良好生物适应性的 SCHOTT 8625 玻璃制作而成，基本不会与人体发生生物排斥现象。其内置的通信芯片，能把近场通信 (NFC) 和射频识别很好地结合在一起，能使用手机上的近距离通信标准，从而实现解锁、传输名片等作用；与此同时，它也支持 ISO14443A 协议的射频识别数据传输，可以用于刷门禁、启动汽车等。摩托罗拉公司目前也在研制拥有同样功能的药片，用户吞下这种药片，即可省却记住密码的烦恼 (图 13.32)。

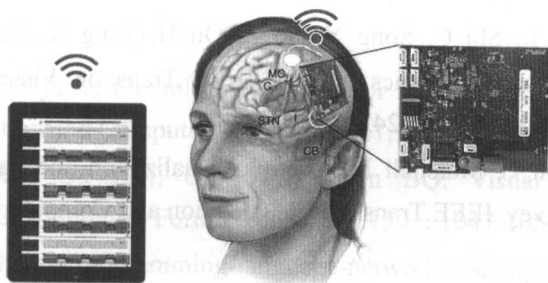


图 13.32 可植入设备工作示意图

可植入设备的研究已经不仅仅局限于医疗领域，但是由于可植入设备的电极、传感器等材料可能被身体组织吸附产生的副作用，医疗领域的皮下植入技术操作规范能否在可植入设备大量普及的情况下得到严格执行，设备能否长时间供电等大量伦理和技术问题都有待研究和探讨，因此可植入设备的发展还有很长的路要走。

13.3 练习题

1. 数据可视化包括哪些内容？
2. 体感互动技术分为哪几类？
3. 什么是增强现实技术？增强现实技术的特点是什么？
4. 当前的可穿戴设备主要有哪几类？

参考文献

- [1] 任磊, 杜一, 马帅, 张小龙, 戴国忠. 大数据可视分析综述. 软件学报, 2014, 25 (9): 1909-1936. <http://www.jos.org.cn/1000-9825/4645.html>
- [2] Collins C, Carpendale S, Penn G. DocuBurst: Visualizing document content using language structure. Computer Graphics Forum, 2009, 28 (3): 1039-1046.
- [3] Han J, Pei J, Yin Y, et al. Mining frequent patterns without candidate generation: a frequent-pattern tree approach. Data Mining and Knowledge Discovery, 2004, 8 (1): 53-87.
- [4] Bongshin Lee Riche, et al. SparkClouds: Visualizing Trends in Tag Clouds. IEEE Transactions on Visualization and Computer Graphics, (Volume: 16, Issue: 6) Nov.-Dec. 2010.
- [5] Havre S, Hetzler E, Whitney P, Nowell L. Themeriver: Visualizing thematic changes in large document collections. IEEE Trans. on Visualization and Computer Graphics, 2002, 8 (1): 9-20.

- [6] Cui W, Liu S, Tan L, Shi C, Song Y, Gao Z, Qu H, Tong X. TextFlow: Towards better understanding of evolving topics in text. *IEEE Trans. on Visualization and Computer Graphics*, 2011, 17 (12) :2412-2421 .
- [7] Herman I, Melancon G, Marshall MS. Graph visualization and navigation in information visualization: A survey. *IEEE Trans. on Visualization and Computer Graphics*, 2000, 6 (1): 24-43 .
- [8] Gou L, Zhang X. Treenetviz: Revealing patterns of networks over tree structures. *IEEE Trans. on Visualization and Computer Graphics*, 2011, 17 (12) :2449-2458 .
- [9] Cui W, Zhou H, Qu H, Wong PC, Li X. Geometry-Based edge clustering for graph visualization. *IEEE Trans. on Visualization and Computer Graphics*, 2008, 1 (6) : 1277-1284 .
- [10] Abello J, van Ham F, Krishnan N. ASK-Graphview: A large scale graph visualization system. *IEEE Trans. on Visualization and Computer Graphics*, 2006, 12 (5) :669-676 .
- [11] Tobler W. Experiments in migration mapping by computer. *The American Cartographer*, 1987, 14 (2) :155-163 .
- [12] Scheepens R, Willems N, Van de Wetering H, Andrienko G, Andrienko N, van Wijk JJ. Composite density maps for multivariate trajectories. *IEEE Trans. on Visualization and Computer Graphics*, 2011, 17 (12) :2518-2527 .
- [13] Peuquet DJ, Kraak MJ. Geobrowsing: Creative thinking and knowledge discovery using geographic visualization. *Information Visualization*, 2002, 1 (1): 80-91 .
- [14] Rhyne TM, MacEachren AM, Dykes J. Exploring geovisualization. *IEEE Computer Graphics and Applications*, 2006, 26 (4) :20-21 .
- [15] Tominski C, Schumann H, Andrienko G, Andrienko N. Stacking-Based visualization of trajectory attribute data. *IEEE Trans. on Visualization and Computer Graphics*, 2012, 18 (12) : 2565-2574 .
- [16] Ahlberg C, Shneiderman B. Visual information seeking: Tight coupling of dynamic query filters with starfield displays. In: Beth A, Susan D, Judith O, eds. *Proc. of the CHI*. New York: ACM Press, 1994. 313-317 .
- [17] Elmqvist N, Draquevic P, Fekete JD. Rolling the dice: Multidimensional visual exploration using Scatterplot matrix navigation. *IEEE Trans. on Visualization and Computer Graphics*, 2008, 14 (6) :1539-1548 .
- [18] Yang J, Hubball D, Ward MS, Rundensterner EA, Ribarsky W. Value and relation display: Interactive visual exploration of large data sets with hundreds of dimensions. *IEEE Trans. on Visualization and Computer Graphics*, 2007, 13 (3) :494-507 .
- [19] Inselberg A, Dimsdale B. Parallel coordinates: A tool for visualizing multi-dimensional geometry. In: Kaufman A, ed. *Proc. of the Visualization*. San Francisco: IEEE Press, 1990.

361-378.

- [20] Claessen JHT, van Wijk JJ. Flexible linked axes for multivariate data visualization. IEEE Trans. on Visualization and Computer Graphics, 2011, 17 (12): 2310-2316.
- [21] Zhou H, Yuan XR, Qu HM, Cui WW, Chen BQ. Visual clustering in parallel coordinates. Computer Graphics Forum, 2008, 27 (3): 1047-1054.
- [22] <http://fatiherikli.github.io/programming-language-network/#language:Java>
- [23] <http://www.visualizing.org/full-screen/406477/embedlaunch>
- [24] 赵勇, 林辉, 沈寓实, 等. 大数据革命——理论、模式与技术创新. 北京: 电子工业出版社, 2014.

第14章

大数据安全与隐私

大数据时代给了人们前所未有的数据采集、存储和处理能力。每一个人都可以把文档、图片、视频等放在云端，享受随时随地同步和查看的便捷性；企业可以将生产、运营、营销、和客户等各个环节数字化，还可以收集全行业的信息，通过移动终端就可以轻松地获得企业生产经营的各种报表和趋势预测；政府的服务和社会化管理则可以通过互联网到达每家每户和每个企业。强大的云数据中心和先进的移动互联网技术使得谷歌眼镜、智能手环这样的可穿戴设备及各种多媒体社交工具盛行，发布信息和检索信息都只在眼睛一眨、指头一动间完成，甚至无须做动作就完成了。但是同时，由于大数据的社会化属性，人们在网络空间的任何数据都可能被收集，人们的资料可能被黑客窃取，人们的朋友圈在社交网络上了一目了然，人们的言论在微博上历历在目，人们的交易和浏览信息随意地被电商挖掘。大数据和云计算就是一把双刃剑，在方便人们生活的同时，安全和隐私问题也日益凸显。

14.1 云计算时代安全与隐私问题凸显

随着数据中心不断整合以及虚拟化、VDI、云端运算应用程序的兴起，越来越多的运算效能与数据都集中到数据中心和服务服务器上。不论是个人信息存储在云盘、邮箱，还是企业将数据存储在云端或使用云计算服务，这些都需要安全保护，安全和隐私问题可以说是云计算和大数据时代所面临的最为严峻的挑战。在 IDC 的一项关于“您认为云计算模式的挑战和问题是什么”的调查中，安全以 74.6% 的比例位居榜首，全球 51% 的首席信息官认

为安全问题是部署云计算时最大的顾虑。趋势科技首席执行官陈怡桦认为：“云计算的日益普及已经使越来越多的云计算服务商进入市场。随着在云计算环境中存储数据的公司越来越多，信息安全问题成为大多数 IT 专业人士最头疼的事情。事实上，数据安全已经是考虑采用云基础设施的机构主要关注的问题之一。”

大数据由于数据集中、目标大，在网络上更容易被盯上；在线数据越来越多，黑客们的犯罪动机也比以往任何时候更强烈；大数据意味着若攻击者成功实施一次攻击，其能得到更多的信息和价值。这些特点都使得大数据更易成为被攻击的目标。

关于网络信息安全，最知名的事件莫过于“棱镜门”了。据美国中情局前职员爱德华·斯诺登披露，“棱镜计划”是一项由美国国家安全局（NSA）于 2007 年小布什时期开始实施的绝密电子监听计划。该计划能够直接进入美国国际网络公司的中心服务器挖掘数据、收集情报，包括微软、雅虎、谷歌、苹果等在内的 9 家国际巨头公司都参与其中，从音频、视频、图片、文档、邮件和链接信息中分析个人的联系方式和行为。

与此同时，公民的隐私泄露事件也层出不穷，这些泄露大部分是黑客攻击企业数据库造成的。据隐私专业公司 PRC（Privacy Rights Clearinghouse）报告称，按保守估计，2011 年全球发生了超过 500 起重大数字安全事故。例如，2011 年 4 月索尼公司由于系统泄露导致 7700 万名用户资料遭窃，遭受了 1.7 亿美元左右的损失；2011 年 12 月，CSDN 的安全系统遭到黑客攻击，600 万名用户的登录名、密码和邮箱遭到泄露；LinkedIn 在 2012 年被曝 650 万名用户账户密码泄露；雅虎遭到网络攻击，致使 45 万名用户 ID 泄露。

另外一些隐私泄露是因为企业产品功能不完善无意造成的。比如几年前，腾讯 QQ 曾经推出朋友圈功能，很多用户的真实名字出现在朋友圈中，引起了用户的强烈抗议，最后腾讯关闭了这一功能。腾讯 QQ 用户真实姓名能在朋友圈中曝光，就是采用了大数据关联分析。由此可见，在大数据搜集和数据分析过程中，随时可能触及用户的隐私，一旦某一环节存在安全隐患，后果不堪设想。

还有一些则是用户个人不注意造成的隐私泄露。比如，有些用户喜欢在 Twitter 等社交网站上发布自己的位置和动态信息，结果有几家网站，如“Please RobMe.com”、“We Know Your House”等，能够根据用户所发的信息，推测出用户不在家的时间，找到用户准确的家庭地址，甚至找出房子的照片。这些网站的做法旨在提醒大家，我们随时暴露在公众视野下，如果不培养安全意识和隐私意识，将会给自身带来灾难。

大数据可以光明正大地搜集用户数据，并可以对用户数据进行分析，这无疑让用户隐私没有任何保障。作为一项新兴的技术，全球很多国家都没有对大数据采集、分析环节进行相应的监管。在没有标准和相应监管措施的情况下，大数据泄露事件频繁发生，已经暴露出大数据时代用户隐私安全的尖锐问题。

当然，我们强调安全和隐私问题，并不是说要因噎废食。正如当今的银行系统，同样存在安全隐患和随时被网络攻击的风险，但是大多数人还是选择把钱存在银行，因为银行的服务为我们提供了便利，同时在绝大多数情况下还是具备安全保障的。我们需要在高效利用云计算和大数据技术的同时，增强安全隐私意识，加强安全防护手段，明确数据归属

及访问权限，完善数据与隐私方面的法规政策等，扎实做好全方位的安全隐私防护，让新技术更好地为我们的生活服务。

14.2 云计算与大数据时代的安全挑战

14.2.1 大数据时代的安全需求

在大数据条件下，越来越多的信息存储在云端，越来越多的服务来自云端，基于公有云的网络信息交互环境带来了与传统条件下不同的安全需求。

1. 机密性

为了保护数据的隐私，数据在云端应该以密文形式存放，但是如果操作不能在密文上进行，那么用户的任何操作都要把涉及的数据密文发送回用户方解密之后再行，将会严重降低效率，因此要以尽可能小的计算开销带来可靠的数据机密性。实现机密性的要求有以下几种情况：一是为了保护用户行为信息的隐私，云服务器要保证用户匿名使用云资源和安全记录数据起源；二是在某些应用情况下，服务器需要在用户数据上面进行运算，而运算结果也以密文形式返回给用户，因此需要使服务器能够在密文上面直接进行操作；三是信息检索是云计算中一个很常用的操作，因此支持搜索的加密是云安全的一个重要需求，但当前已有的支持搜索的加密只支持单关键字搜索，所以支持多关键字搜索、搜索结果排序和模糊搜索是云计算的另一需求方向。

2. 数据完整性

在基于云的存储服务，如 Amazon 简单存储服务 S3、Amazon 弹性块存储 EBS，以及 Nirvanix 云存储服务中，必须要保证数据存储的完整性。在云存储条件下，因为可能面临软件失效或硬件损坏导致的数据丢失、云中其他用户的恶意损坏、服务商为经济利益擅自删除一些不常用数据等情况，用户无法完全相信云服务器会对自己的数据进行完整性保护，所以用户需要对其数据的完整性进行验证。这就需要系统提供远程数据完整性验证和数据恢复功能。

3. 访问控制

云计算中要阻止非法的用户对其他用户的资源和数据的访问，细粒度地控制合法用户的访问权限，因此云服务器需要对用户的访问行为进行有效的验证。其访问控制需求主要包括以下两个方面：一是网络访问控制，指云基础设施中主机之间互相访问的控制；二是数据访问控制，指云端存储的用户数据的访问控制。数据的访问控制中要保证对用户撤销操作、用户动态加入和用户操作可审计等要求的支持。

4. 身份认证

云计算系统应建立统一、集中的认证和授权系统，以满足云计算多租户环境下复杂的用户权限策略管理和海量访问认证要求，提高云计算系统身份管理和认证的安全性。现有的身份认证技术主要包括三类：一是基于用户持有的秘密口令的认证，二是基于用户持有的硬件（如智能卡、U盾等）的认证，三是基于用户生物特征（如指纹）的认证。但是这些方法都是通过某一维度的特征来进行认证的，对重要的隐私信息和商业机密来讲安全性仍不够强。最新提出的层次化的身份认证在多个云之间实现层次化的身份管理，多因子身份认证从多重特征上对客户进行认证，都是身份认证技术的新需求。

5. 可信性

虚拟空间用户与云服务商之间在相互信任的基础上达成协议进行服务，可信性是云计算健康发展的基本保证，也是基本需求。具体包括服务商和用户的可信性两个方面。服务商可信是指其向其他服务商或者用户提供的服务必须是可信的，而不是恶意的；用户可信是指用户采用正常、合法的方式访问服务商提供的服务，用户的行为不会对服务商本身造成破坏。如何实现云计算的问责功能，通过记录操作信息等手段实现对恶意操作的追踪和问责；如何通过可信计算、安全启动、云端网关等技术手段构建可信的云计算平台，达到云计算的可信性都是可信性方面需要研究的问题。

6. 防火墙配置安全性

在基础设施云中的虚拟机需要进行通信，这些通信分为虚拟机之间的通信和虚拟机与外部的通信。通信的控制可以通过防火墙来实现，因此防火墙的配置安全性非常重要。如果防火墙配置出现问题，那么攻击者很可能利用一个未被正确配置的端口对虚拟机进行攻击。因此，在云计算中，需要设计对虚拟机防火墙配置安全性进行审查的算法。

7. 虚拟机安全性

虚拟机技术在构建云服务架构等方面广泛应用，但与此同时，虚拟机也面临着两方面的安全性，一方面是虚拟机监督程序的安全性，另一方面是虚拟机镜像的安全性。在以虚拟化为支撑技术的基础设施云中，虚拟机监督程序是每台物理机上的最高权限软件，因此其安全的重要性毋庸置疑。另外，在使用第三方发布的虚拟机镜像的情况下，虚拟机镜像中是否包含恶意软件、盗版软件等，也是需要进行检测的。

14.2.2 信息安全的发展历程

广义的信息安全涉及各种情报、商业机密、个人隐私等，在各行各业都早已存在。具体到计算机通信领域的信息安全则是最近几十年随着电子信息技术的发展而兴起的。信息安全的发展大致经历了四个时期。

第一个时期是通信安全时期，其主要标志是 1949 年香农发表的《保密通信的信息理论》。在这个时期通信技术还不发达，电脑只是零散地位于不同的地点，信息系统的安全仅限于保证电脑的物理安全，以及通过密码（主要是序列密码）解决通信安全的保密问题。把电脑安置在相对安全的地点，不允许非授权用户接近，就基本可以保证数据的安全性了。这个时期的安全性是指信息的保密性，对安全理论和技术的研究也仅限于密码学。这一阶段的信息安全可以简称为通信安全。它侧重于保证数据从一地传送到另一地时的安全性。

第二个时期为计算机安全时期，以 20 世纪 70~80 年代的可信计算机系统评价准则（TCSEC）为标志。20 世纪 60 年代以后，半导体和集成电路技术的飞速发展推动了计算机软、硬件的发展，计算机和网络技术的应用进入了实用化和规模化阶段，数据的传输已经可以通过计算机网络来完成。这时候的信息已经分成静态信息和动态信息。人们对安全的关注已经逐渐扩展为以保密性、完整性和可用性为目标的信息安全阶段，主要保证动态信息在传输过程中不被窃取，即使窃取了也不能读出正确的信息；还要保证数据在传输过程中不被篡改，让读取信息的人能够看到正确无误的信息。1977 年美国国家标准局（NBS）公布的国家数据加密标准（DES）和 1983 年美国国防部公布的可信计算机系统评价准则（Trusted Computer System Evaluation Criteria, TCSEC，俗称橘皮书，1985 年再版）标志着解决计算机信息系统保密性问题的研究和应用迈上了历史的新台阶。

第三个时期是在 20 世纪 90 年代的网络时代。从 20 世纪 90 年代开始，由于互联网技术的飞速发展，无论是企业内部信息还是外部信息都得到了极大的开放，而由此产生的信息安全问题跨越了时间和空间，信息安全的焦点已经从传统的保密性、完整性和可用性三个原则发展为诸如可控性、抗抵赖性、真实性等其他的原则和目标。

第四个时期是进入 21 世纪的信息安全保障时代，其主要标志是《信息保障技术框架》（IATF）。如果说对信息的保护，主要还是处于从传统安全理念到信息化安全理念的转变过程中，那么面向业务的安全保障，就完全是从信息化的角度来考虑信息的安全了。体系性的安全保障理念，不仅关注系统的漏洞，而且从业务的生命周期着手，对业务流程进行分析，找出流程中的关键控制点，从安全事件出现的前、中、后三个阶段进行安全保障。面向业务的安全保障不是只建立防护屏障，而是建立一个“深度防御体系”，通过更多的技术手段把安全管理与技术防护联系起来，不再是被动地保护自己，而是主动地防御攻击。也就是说，面向业务的安全防护已经从被动走向主动，安全保障理念从风险承受模式走向安全保障模式。信息安全阶段也转化为从整体角度考虑其体系建设的信息安全保障时代。

14.2.3 新兴信息技术带来的安全挑战

物联网、云计算、大数据和移动互联网被称为新一代信息技术的“四驾马车”，它们提供了科技发展的核心动力，在给政府、企业、社会和人民带来极大便利的同时，也催生了不同于以往的安全问题和威胁。在传统的安全防护体系中，“防火墙”起着至关重要的作用。防火墙是一种形象的说法，其实它是一种计算机硬件和软件的组合，在内部网络与外部网

络之间建立起一个安全网关，从而保护内部网络免受外部非法用户的侵入。然而在云计算时代，公有云是为多租户服务的，很多不同用户的应用都运行在同一个云数据中心内，这就打破了传统的安全体系中的内外之分。对于企业和用户来说，不仅要防范来自数据中心外部的攻击，还要提防云服务的提供商，以及潜藏在云数据中心内部的其他别有用心用户，形象地说就是“家贼难防”。这就使得用户与云服务商的信任关系的建立、管理和维护更加困难，同时对用户的服务授权和访问控制也变得更加复杂（图 14.1）。

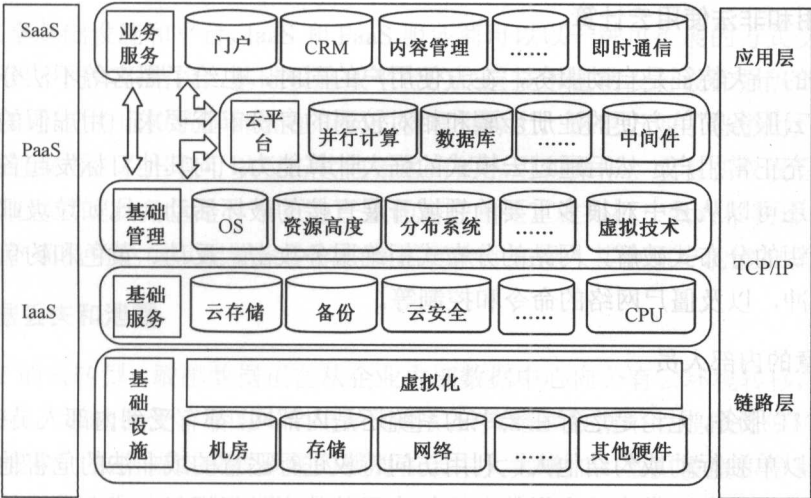


图 14.1 云共享服务模式

现有的安全理论与实践大多针对传统的计算模式，不能完全适用于云计算的新商业模式和技术架构。在安全隐私方面，大部分云计算服务商无法在短期内达到企业内部网的成熟度，更不用说提供比内网更高的安全服务。据调查显示，当前所有云服务商都无法通过完全的合规审计，更无法抵御“坏分子”（黑客和其他犯罪者）的多方攻击，所以任何云服务商都不敢向企业用户提供敏感隐私数据的安全服务等级协议。安全与合规已经成为大多数企业 IT 向云转型的头号顾虑，它们无法放心地将高价值数字资产放入云中。

在云计算时代，“坏分子”进行云攻击的方式有 3 种，见表 14.1。其中侧面攻击是云安全防护的重点对象。

表 14.1 “坏分子”攻击方式

代号	名称	说明	例子
F	前面攻击 (Front Attack)	数据隐私从内部网移到云中后，遭受来自互联网的攻击	黑客利用 Web 应用中的安全漏洞，窃取企业数据库中的信息
S	侧面攻击 (Side Attack)	公有云中的数据隐私，遭受同一云中其他租户的攻击	竞争者利用 CPU Covert Channel 安全缺陷，窃取同一物理机上对方虚拟机的密钥
B	后面攻击 (Back Attack)	云中的数据隐私，遭受来自云服务商内部人员的攻击	不满者利用管理员权限，窃取客户信息

云时代安全攻击的具体方式有很多种分类,根据美国知名市场研究公司 Gartner 发布的“云计算风险评估”研究报告,企业存储在云服务商处的数据,存在 7 种潜在安全风险:特权用户准入风险、法律遵从、数据位置、数据隔离、数据恢复、审计支持和数据长期生存性。ENISA(欧洲网络与信息安全署)提出了一个采用 ISO 27000 系列标准的云计算信息安全保障体系架构,主要涉及的安全风险包括:隐私安全、身份和访问管理、环境安全、法规和物理安全等。总体来说,在业界得到广泛认可的安全风险主要包括以下 8 种类型。

1. 滥用和非法使用云计算

云计算的一大特征是自助服务,在方便用户的同时,也给了黑客等不法分子机会,他们可以利用云服务简单方便的注册步骤和相对较弱的身份审查要求,用虚假的或盗取的信息注册,冒充正常用户,然后通过云模式的强大计算能力,向其他目标发起各种各样的攻击。攻击者还可以从云中很多重要的领域开展直接的破坏活动,比如垃圾邮件的制作传播,用户密钥的分布式破解,网站的分布式拒绝服务攻击,反动、黄色和钓鱼欺诈等不良信息的云缓冲,以及僵尸网络的命令和控制等。

2. 恶意的内部人员

所有的 IT 服务,无论是运行在云中的系统还是内部网,都有受到内部人员破坏的风险。内部人员可以单独行动或勾结他人,利用访问特权进行恶意的或非法的危害他人的行动。内部人员搞破坏的原因是多种多样的,比如为了某件事进行报复,或者发泄他们心中对社会的不满,或者为了物质上的利益等。

在云计算时代,这种威胁对于消费者来说大大增加了。首先,由于云服务商一般拥有大量企业用户,雇佣的 IT 管理人员数量比单独一个企业的 IT 管理人员多得多;其次,云计算也是 IT 服务外包的一种形式,所以也继承了外包服务商的恶意内部人员风险。因此,云计算中的监管不仅在操作上更为困难,而且风险也是个未知数。

3. 不安全的编程接口

云服务商一般都会为用户提供应用程序接口(API),让用户使用、管理和扩展自己的云资源。云服务的流程都要用到这些 API,比如创建虚拟机、管理资源、协调服务、监控应用等。大量的 API 多多少少都会有安全漏洞,有些属于设计缺陷,有些属于代码缺陷。黑客利用软件漏洞,就可以攻击任何用户。

4. 身份或服务账户劫持

身份或服务账户劫持是指在用户不知情或没有批准的情况下,他人恶意地取代用户的身份或劫持其账户。账户劫持的方法包括网络钓鱼、欺骗和利用软件漏洞持续攻击等。

在云时代,这类威胁也变得更为严重。云服务不同于传统的企业,它没有广泛的基于角色或团体接入的权限隔离,通常身份密码被重复使用在很多站点和服务上,同样的内部账户被用于管理软件系统、管理服务器和追踪账单。更加糟糕的是,账户经常在不同用户

间共享。不管对于用户还是管理员，大多数云服务缺乏基础设施和流程去实现强验证。

一旦攻击者获取了用户的身份密码，他们就可以窃听用户的活动和交易，获取和操控数据，发布错误的信息，并将客户导向非法站点。客户的账户或服务还可能变成攻击者的新基地，他们从这里冒用受害者的名义和影响力再去发动新的攻击。他们还可以强制让账户所有者支付无用的 CPU 时间、存储空间或其他被计量付费的资源。

5. 资源隔离问题

通过共享基础设施和平台，IaaS 和 PaaS 服务商可以以一种可扩展的方式交付他们的服务，这种多租户的体系结构、基础设施或平台的底层技术通常没有设计强隔离。资源虚拟化支持将不同租户的虚拟资源部署在相同的物理资源上，这也方便了恶意用户借助共享资源实施侧通道攻击。攻击者可以攻击其他云客户的应用和操作，或者获取没有进行授权访问的数据。取得管理员角色是一个更为严重的潜在危险，虚拟机一般对根物理机很难设防，通过物理机管理员角色可以配置命令和控制恶意软件来侵入其他用户的虚拟机。

6. 数据丢失和泄露

随着 IT 的云转型，敏感数据正在从企业内部数据中心向公有云环境转移，伴随着优点而来的是缺点，那就是云计算的安全隐私问题。云策略和数据中心虚拟化使防卫保护的现实变得更加复杂，数据被盗或被泄露的威胁在云中大大增加。数据被盗和隐私泄露可以对企业和个人产生毁灭性的影响，除了对云服务商品品牌和名声造成损害外，还可能导致关键知识的损失，产生竞争力的下降和财产方面的损失。此外，丢失或泄露的数据可能会遭到破坏和滥用，甚至引起各种法律纠纷。

7. 商业模式变化风险

云计算的一个宗旨是减少用户对硬件和软件的维护工作，使他们可以将精力集中于自己的核心业务。云计算固然有着明显的财政和操作方面的优势，但云服务商必须解除用户对安全的担忧。当用户评估云服务的安全状态时，软件的版本、代码的更新、安全规则、漏洞状态、入侵尝试和安全设计都是重要的影响因素。除了网络入侵日志和其他记录，谁与自己分享基础架构的信息也是用户要知道的。

8. 对企业内部网的攻击

很多企业用户将混合云作为一种减少公有云中风险的方式。混合云是指混合地使用公有云和企业内部网络资源（或私有云）。在这种方案中，客户通常把网页前台移到公有云中，而把后台数据库留在内部网络中。在云和内部网络之间，一个虚拟或专用的网络通道被建立起来，这就开启了对企业内部网络攻击的机会，导致本来被安全边界和防火墙保护的企业内部网络随时可能受到来自云的攻击。但如果这一通道关闭，由混合云支持的业务将被停止，将会给企业带来重大的财产损失。

这里还要特别提到个人设备安全管理。随着移动互联网和大数据的快速发展，移动设

备的应用也在不断增长。随着 BYOD（携带自己的设备办公）风潮的普及，许多企业开始考虑允许员工自带智能设备使用企业内部应用，其目标是在满足员工自身追求新科技和个性化的同时提高工作效率，降低企业成本。然而，这样做带来的风险也是很大的，员工带着自己的设备连接企业网络，就可能让各种木马病毒或恶意软件到处传播，造成安全隐患。

14.3 如何解决安全问题

云计算和大数据的新商业模式和技术架构在带给人类更多经济、方便、快捷、智能化体验的同时，也给信息安全和个人隐私带来了全新的威胁。要促进云计算和大数据技术的健康发展，就必须直面安全和隐私问题，而这需要大量的实践研究工作。同时，云计算安全并不仅仅是技术问题，它还涉及标准化、监管模式、法律法规等诸多方面。因此，仅从技术角度出发探索解决云计算安全问题是远远不够的，还需要信息安全学术界、产业界以及政府相关部门的共同努力。

2008 年成立的云安全联盟（Cloud Security Alliance, CSA），就是在安全隐私将云逼得走投无路的时候应运而生的世界性的行业组织。CSA 总部位于云计算之都西雅图，微软和亚马逊的总部也在这里。CSA 任命世界顶级安全专家出任其最重要的首席研究官，大力开展实践安全研究，在厂商指导、用户培训、政府协调和高校合作等各方面也起着举足轻重的纽带作用。目前参与云安全联盟并接受指导的会员厂商有上百家，包括微软、亚马逊、谷歌、英特尔、甲骨文、赛门铁克、华为等全球云计算领军企业。

云端厂商在云安全方面的努力不言而喻，比如微软的云计算数据中心、云平台和 Office 365 等多项云服务都获得了多个国家政府和行业组织的安全认证，采取了政府和企业级安全保护措施。

用户在云转型中的努力也非常重要，特别是要有敢于承担风险的精神。用户在云部署过程中要识别数字资产，将资产映射到可能的云部署模型中，然后评估云中风险。美国政府 IT 部门都大量采用云服务，它们是用户云转型的典型范例。

各国政府需要出台鼓励云计算的政策、法规、标准和战略。美国政府已经制定了云计算的一些标准，比如美国技术标准局创立了云计算模型和云参考架构；欧盟正在 CSA 的帮助下起草云计算战略；中国云计算安全政策与法规工作组也发表了蓝皮书。

高等院校则需要大力培养云安全人才，为云计算的长久发展输送新鲜血液。美国华盛顿大学的赛博安全中心已率先开启了研究生的云安全实践研究项目，其将在美国国防部、国安部和国家科学基金会的支持下，由 CSA 导师指导进行研究。

基于当前的研究成果，解决云安全问题主要有两类途径：

- ① 建立完善的安全防护框架，加强云安全技术研究；
- ② 创立本质安全的新信息技术基础。

14.3.1 云计算安全防护框架

解决云计算安全问题的当务之急是针对威胁，建立综合性的云计算安全防护框架，并积极开展其中各个云安全的关键技术研究。遵循共同的安全防护框架是为了消除广大用户（特别是政府和企业）所承担的风险，明确各机构的义务，避免漏洞，实现完整有效的安全防护措施。当前业界知名的防护框架有美国国家技术标准局（NIST）防护框架、CSA 防护框架等。

（1）NIST 防护框架涵盖的领域如下

① 治理（Governance）。各机构在应用开发和服务提供中采用的现有良好实践措施需要延伸到云中。这些实践要继续遵从机构相应的政策、程序和标准，用于在云中的设计、实施、测试、部署和监测。审计机制和工具需要到位，以确保机构的实践措施在整个系统的生命周期内都有效。

② 合规（Compliance）。用户要了解各类和安全隐私相关的法律和规章制度以及自己机构的义务，特别是那些涉及存放位置的数据、隐私和安全控制及电子证据发现的要求。用户要审查和评估云服务提供商的产品，并确保合同条款充分满足法规要求。

③ 信任（Trust）。安全和隐私保护措施（包括能见度）需要纳入云计算服务合同中，并建立具有足够灵活性的风险管理制度，以适应不断发展和不断变化的风险状况。

④ 架构（Architecture）。用户要了解云服务提供商的底层技术和管理技术，包括设计安全的技术控制和对隐私的影响，了解系统完整的生命周期及其系统组件。

⑤ 身份和访问管理（Identity and Access Management）。云服务提供商要确保有足够的保障措施，能够安全地实行认证、授权和提供其他身份及访问管理功能。

⑥ 软件隔离（Software Isolation）。用户要了解云服务提供商采用的虚拟化和其他软件隔离技术，并评估所涉及的风险。

⑦ 数据保护（Data Protection）。用户要评估云服务提供商的数据管理解决方案的适用性，确定能否消除托管数据的顾虑。

⑧ 可用性（Availability）。云服务提供商要确保在中期或长期中断或严重的灾难时，关键运营操作可以立即恢复，最终所有运营操作都能够及时地和有条理地恢复。

⑨ 应急响应（Incident Response）。用户要向云服务提供商了解和洽谈合同中涉及事件应急响应和处理的程序，以满足自己组织的要求。

（2）CSA 防护框架涉及的领域如下

① 合规（Compliance）：见 NIST 同名领域。

② 数据治理（Data Governance）：见 NIST 同名领域。

③ 设施安全（Facility Security）：即云数据中心的物理安全。

④ 人事安全（Human Resources Security）：包括云服务商员工的聘用合同及备件调查等。

- ⑤ 信息安全 (Information Security): 即信息技术安全防护控制。
- ⑥ 法律 (Legal): 指云服务应遵守的各国法律法规等。
- ⑦ 运营管理 (Operations Management): 云服务商系统及员工的运营管理和监控。
- ⑧ 风险管理 (Risk management): 包括云计算的风险识别、评估和管理。
- ⑨ 发布管理 (Release Management): 服务发布和改变的管理。
- ⑩ 恢复性 (Resiliency): 包括对事故和灾难的恢复能力。
- ⑪ 安全架构 (Security Architecture): 即云计算的安全设计。

在业界提出的这些防护框架的基础上,冯登国、张敏等人在《云计算安全研究》中提出了一种包括云计算安全服务体系与云计算安全标准及测评体系两大部分的云安全框架建议。

1. 云计算安全服务体系

云计算安全服务体系由一系列云安全服务构成,是实现云用户安全目标的重要技术手段。根据其所属层次的不同,云安全服务可以进一步分为云基础设施服务、云安全基础服务以及云安全应用服务3类。

(1) 云基础设施服务

云基础设施服务为上层云应用提供安全的数据存储、计算等IT资源服务,是整个云计算体系安全的基石。这里,安全性包含两个层面的含义:一是抵挡来自外部黑客的安全攻击的能力,二是证明自己无法破坏用户数据与应用的能力。一方面,云平台应分析传统计算平台面临的安全问题,采取严密的安全措施。例如,在物理层考虑厂房安全,在存储层考虑完整性和文件/日志管理、数据加密、备份、灾难恢复等,在网络层考虑拒绝服务攻击、DNS安全、网络可达性、数据传输机密性等,系统层则应涵盖虚拟机安全、补丁管理、系统用户身份管理等安全问题,数据层包括数据库安全、数据的隐私性与访问控制、数据备份与清洁等,而应用层应考虑程序完整性检验与漏洞管理等。另一方面,云平台应向用户证明自己具备某种程度的数据隐私保护能力。例如,存储服务中证明用户数据以密态形式保存,计算服务中证明用户代码运行在受保护的内存中,等等。由于用户安全需求方面存在着差异,云平台应具备提供不同安全等级的云基础设施服务的能力。

(2) 云安全基础服务

云安全基础服务属于云基础软件服务层,为各类云应用提供共性信息安全服务,是支撑云应用满足用户安全目标的重要手段。其中比较典型的云安全服务包括以下几种。

① 云用户身份管理服务。主要涉及身份的供应、注销以及身份认证过程。在云环境下,实现身份联合和单点登录可以支持云中合作企业之间更加方便地共享用户身份信息和认证服务,并减少重复认证带来的运行开销。但云身份联合管理过程应在保证用户数字身份隐私性的前提下进行。由于数字身份信息可能在多个组织间共享,其生命周期各个阶段的安全性管理更具有挑战性,而基于联合身份的认证过程在云计算环境下也具有更高的安全需求。

② 云访问控制服务。云访问控制服务的实现依赖于妥善地将传统的访问控制模型（如基于角色的访问控制模型、基于属性的访问控制模型以及强制/自主访问控制模型等）和各种授权策略语言标准（如 XACML、SAML 等）扩展后移植入云环境。此外，鉴于云中各企业组织提供的资源服务兼容性和可组合性的日益提高，组合授权问题也是云访问控制服务安全框架需要考虑的重要问题。

③ 云审计服务。由于用户缺乏安全管理与举证能力，要明确安全事故责任就要求服务商提供必要的支持。因此，由第三方实施的审计就显得尤为重要。云审计服务必须提供满足审计事件列表的所有证据以及证据的可信度说明。当然，若要该证据不会披露其他用户的信息，则需要特殊设计的数据取证方法。此外，云审计服务也是保证云服务商满足各种合规性要求的重要方式。

④ 云密码服务。由于云用户中普遍存在数据加、解密运算需求，云密码服务的出现也是十分自然的。除最典型的加、解密算法服务外，密码运算中密钥管理与分发、证书管理及分发等都可以基础类云安全服务的形式存在。云密码服务不仅为用户简化了密码模块的设计与实施，也使得密码技术的使用更集中、规范，也更易于管理。

（3）云安全应用服务

云安全应用服务与用户的需求紧密结合，种类繁多。典型的例子，如 DDoS 攻击防护云服务、Botnet 检测与监控云服务、云网页过滤与杀毒应用、内容安全云服务、安全事件监控与预警云服务、云垃圾邮件过滤及防治等。传统网络安全技术在防御能力、响应速度、系统规模等方面存在限制，难以满足日益复杂的安全需求，而云计算优势可以极大地弥补上述不足。云计算提供的超大规模计算能力与海量存储能力，能在安全事件采集、关联分析、病毒防范等方面实现性能的大幅提升，可用于构建超大规模安全事件信息处理平台，提升全网安全态势把握能力。此外，还可以通过海量终端的分布式处理能力进行安全事件采集，上传到云安全中心分析，极大地提高了安全事件搜集与及时处理的能力。

2. 云计算安全标准及测评体系

云计算安全标准及测评体系为云计算安全服务体系提供了重要的技术与管理支撑，其核心至少应涵盖以下几方面内容。

① 云服务安全目标的定义、度量及其测评方法规范。该规范帮助云用户清晰地表达其安全需求，并量化其所属资产各安全属性指标。清晰而无二义的安全目标是解决服务安全质量争议的基础。这些安全指标具有可测量性，可通过指定测评机构或者第三方实验室测试评估。规范还应指定相应的测评方法，通过具体操作步骤检验服务提供商对用户安全目标的满足程度。由于在云计算中存在多级服务委托关系，相关测评方法仍有待探索实现。

② 云安全服务功能及其符合性测试方法规范。该规范定义基础性的云安全服务，如云身份管理、云访问控制、云审计以及云密码服务等的主要功能与性能指标，便于使用者在选择时对比分析。该规范将起到与当前 CC 标准中的保护轮廓（PP）与安全目标（ST）类似的作用。而判断某个服务商是否满足其所声称的安全功能标准需要通过安全测评，需要

与之相配合的符合性测试方法与规范。

③ 云服务安全等级划分及测评规范。该规范通过云服务的安全等级划分与评定，帮助用户全面了解服务的可信程度，更加准确地选择自己所需的服务。尤其是底层的云基础设施服务以及云基础软件服务，其安全等级评定的意义尤为突出。同样，验证服务是否达到某安全等级需要相应的测评方法和标准化程序。

14.3.2 基础云安全防护关键技术

建立完善的云安全防护框架可以从顶层设计上实现安全防护的全方位、无漏洞，要实现云安全防护，关键还是要有针对性地进行相关技术的研究。对于前面攻击，传统的网络安全和应用安全防护手段如身份认证、防火墙、入侵监测、漏洞扫描等仍然适合。而对于侧面攻击和后面攻击，可以考虑从云服务模式和数据保护两个角度分别采取不同的防护手段。

从数据保护的角度，可以采取表 14.2 中的防护手段。

表 14.2 数据保护角度的防护手段

数据保护	对服务提供商	对其他用户
数据访问的权限控制	存储加密	存储隔离
数据运行时的私密性	操作系统隔离	虚拟机隔离、操作系统隔离
数据传输私密性	网络加密	传输层加密
数据完整性	数据检验	
持久可用性	数据备份、镜像、分布式存储	

云服务从服务模式上分为基础设施即服务（IaaS）层、平台即服务（PaaS）层和软件即服务（SaaS）层三层，对于不同服务模式存在的安全风险可以采取表 14.3 中的防护手段。

表 14.3 云安全防护手段

软件即服务层	用户身份认证与访问管理	对用户身份进行认证，设置用户权限，进行访问管理
	信任及服务授权体系	信任服务体系：在用户的实体和网络空间中的用户角色之间建立一种映射关系，将现实的物理世界中的信任关系移植到虚拟的网络空间中 授权服务体系：主要是为网络空间提供用户操作授权的管理，在网络空间中的用户角色与最终应用系统中用户的操作权限之间建立一种映射关系
	动态访问控制策略	根据实际情况动态创建可访问列表，扩展性和安全性更强
	终端安全策略	使终端“可信、可控、可管、可用”，保证终端正常、安全地运行

续表

平台即服务层	平台安全性验证	保障平台中的应用及数据安全
	服务安全性验证	保障服务安全、可靠、可用
	服务审计与监管	保持完备的日志及记录
	数据完整性验证	保障数据不丢失、不被篡改和破坏
	可计算加密技术	在保障数据加密的状态下进行计算
基础设施即服务层	虚拟化软件安全	保障虚拟化管理软件本身的安全性
	虚拟化网络安全	保障虚拟网络的安全性、隔离性
	虚拟化存储安全	保障虚拟存储的安全性、隔离性、可用性
	虚拟机镜像安全	保障虚拟机镜像的安全

在当前的云计算安全防护技术中，比较热门的研究主要有以下几种。

1. 可信访问控制

由于无法信赖服务商忠实实施用户定义的访问控制策略，所以在云计算模式下，大家更加关心的是如何通过非传统访问控制类手段实施数据对象的访问控制。其中得到关注最多的是基于密码学方法实现访问控制，包括：基于层次密钥生成与分配策略实施访问控制的方法；利用基于属性的加密算法[如密钥规则的基于属性加密方案（KP-ABE），或密文规则的基于属性加密方案（CP-ABE）]，基于代理重加密的方法；以及在用户密钥或密文中嵌入访问控制树的方法等。基于密码类方案面临的一个重要问题是权限撤销。一个基本方案是为密钥设置失效时间，每隔一定时间，用户从认证中心更新私钥；另外就是基于用户的唯一 ID 属性及非门结构，实现对特定用户进行权限撤销。但目前看来，上述方法在带有时间或约束的授权、权限受限委托等方面仍存在许多有待解决的问题。

2. 密文检索与处理

数据变成密文时丧失了许多其他特性，导致大多数数据分析方法失效。密文检索有两种典型的方法：基于安全索引的方法通过为密文关键词建立安全索引，检索索引查询关键词是否存在；基于密文扫描的方法对密文中的每个单词进行比对，确认关键词是否存在，以及统计其出现的次数。密文处理研究主要集中在秘密同态加密算法设计上。早在 20 世纪 80 年代，就有人提出多种加法同态或乘法同态算法，但是由于被证明安全性存在缺陷，后续工作基本处于停顿状态。而近期，IBM 研究员 Gentry 利用“理想格（Ideal Lattice）”的数学对象构造隐私同态（Privacy Homomorphism）算法，或称全同态加密，使人们可以充分地操作加密状态的数据，在理论上取得了一定突破，使相关研究重新得到研究者的关注，但目前与实用化仍有很长的距离。

3. 数据存在与可使用性证明

由于大规模数据所导致的巨大通信代价，用户不可能将数据下载后再验证其正确性。因此，云用户需要在取回很少数据的情况下，通过某种知识证明协议或概率分析手段，以

高置信概率判断远端数据是否完整。典型的工作包括：面向用户单独验证的数据可检索性证明(POR)方法、公开可验证的数据持有证明(PDP)方法。NEC 实验室提出的 PDI(Provable Data Integrity) 方法改进并提高了 POR 方法的处理速度以及验证对象规模, 且能够支持公开验证。其他典型的验证技术包括: Yun 等人提出的基于新的树形结构 MAC Tree 的方案, Schwarz 等人提出的基于代数签名的方法, Wang 等人提出的基于 BLS 同态签名和 RS 纠错码的方法等。

4. 数据隐私保护

云中数据隐私保护涉及数据生命周期的每一个阶段。Roy 等人将集中信息流控制(DIFC)和差分隐私保护技术融入云中的数据生成与计算阶段, 提出了一种隐私保护系统 airavat, 可防止 MapReduce 计算过程中非授权的隐私数据泄露出去, 并支持对计算结果的自动除密。在数据存储和使用阶段, Mowbray 等人提出了一种基于客户端的隐私管理工具, 提供以用户为中心的信任模型, 帮助用户控制自己的敏感信息在云端的存储和使用。Munts-Mulero 等人讨论了现有的隐私处理技术, 包括 K 匿名、图匿名以及数据预处理等。Rankova 等人则提出了一种匿名数据搜索引擎, 可以使交互双方搜索对方的数据, 获取自己所需要的部分, 同时保证搜索询问的内容不被对方所知, 搜索时与请求不相关的内容不会被获取。

5. 虚拟安全技术

虚拟技术是实现云计算的关键核心技术, 使用虚拟技术的云计算平台上的云架构提供者必须向其客户提供安全性和隔离保证。Santhanam 等人提出了基于虚拟机技术实现的 grid 环境下的隔离执行机。Raj 等人提出了通过缓存层次可感知的核心分配, 以及基于缓存划分的页染色的两种资源管理方法, 实现性能与安全隔离。这些方法在隔离影响一个 VM 的缓存接口时是有效的, 并被整合到一个样例云架构的资源管理(RM)框架中。

6. 云资源访问控制

在云计算环境中, 各个云应用属于不同的安全域, 每个安全域都管理着本地的资源和用户。当用户跨域访问资源时, 需要在域边界设置认证服务, 对访问共享资源的用户进行统一的身份认证管理。在跨多个域的资源访问中, 各域有自己的访问控制策略, 在进行资源共享和保护时必须对共享资源制定一个公共的、双方都认同的访问控制策略, 因此, 需要支持策略的合成。这个问题最早由 Mclean 在强制访问控制框架下提出, 他提出了一个强制访问控制策略的合成框架, 将两个安全格合成一个新的格结构。策略合成的同时还要保证新策略的安全性, 新的合成策略不能违背各个域原来的访问控制策略。为此, Gong 提出了自治原则和安全原则。Bonatti 提出了一个访问控制策略合成代数, 基于集合论使用合成运算符来合成安全策略。Wijesekera 等人提出了基于授权状态变化的策略合成代数框架。Agarwal 构造了语义 Web 服务的策略合成方案。Shafiq 提出了一个多信任域 RBAC 策略合成策略, 侧重于解决合成的策略与各域原有策略的一致性问题的。

7. 可信云计算

将可信计算技术融入云计算环境,以可信赖方式提供云服务已成为云安全研究领域的一大热点。Santos 等人提出了一种可信云计算平台 TCCP,基于此平台,IaaS 服务商可以向其用户提供一个密闭的箱式执行环境,保证客户虚拟机运行的机密性。另外,它允许用户在启动虚拟机前检验 IaaS 服务商的服务是否安全。Sadeghi 等人认为,可信计算技术提供了可信的软件和硬件以及证明自身行为可信的机制,可以被用来解决外包数据的机密性和完整性问题;同时设计了一种可信软件令牌,将其与一个安全功能验证模块相互绑定,以求在不泄露任何信息的前提条件下,对外包的敏感(加密)数据执行各种功能操作。

14.3.3 创立本质安全的新型 IT 体系

当前计算机和互联网的安全措施都是被动和暂时的,普通用户被迫承担安全责任,频繁地扫描漏洞和下载补丁。进入云计算时代,不少厂商适时推出云安全和云杀毒产品,可以想象,云病毒和云黑客们的水平必然有所提高。

实际上,今天遭遇信息和网络安全的根源,在于当初发明计算机和网络时根本没想到用户中有恶意的攻击者,或者说没有预见到安全隐患。PC 时代的防火墙和杀毒软件,以及各种法律法规,只能通过事后补救来处罚给他人利益造成损害的人。这些措施不能满足社会信息中枢的可控开放模式和安全需求。其实,抓住云计算的机遇,重新规划计算机和互联网基础理论,建立完善的安全体系并不困难。

下面我们将在分析 IP 互联网安全问题原因的基础上,提出大一统网络根治网络安全的一揽子解决方案。网络安全不是一项可有可无的服务,大一统网络的目标不是用复杂设备和多变的软件来改善网络安全性,而是直接建立本质上高枕无忧的网络。

从网络地址结构上根治仿冒。IP 互联网的地址由用户设备告诉网络,大一统网络地址由网络告诉用户设备。

为了防范他人入侵,PC 和互联网设置了烦琐的口令、密码障碍。就算是实名地址,仍无法避免密码被破译或由于用户的失误而造成的安全信息泄露。连接到 IP 互联网上的 PC 终端,首先必须自报家门,告诉网络自己的 IP 地址,但网络却无法保证这个 IP 地址的真假。这就是 IP 互联网第一个无法克服的安全漏洞。

大一统网络终端的地址是通过网管协议生成的,用户终端只能用这个生成的地址进入网络,因此无须认证,确保不会错。大一统网络地址不仅具备唯一性,而且具备可定位和可定性功能,如同个人身份证号码一样,隐含了该用户端口的地理位置、设备性质和服务权限等其他特征。交换机根据这些特征规定了分组包的行为规则,实现不同性质的数据分流。

每次服务发放独立通行证,阻断黑客攻击的途径。IP 互联网可以自由进出,用户自备防火墙,大一统网络每次服务必须申请通行证。

IP 通信协议在用户终端执行，这就可能被篡改。路由信息在网上传播，这就可能被窃听。网络中的固有缺陷导致了地址欺骗、匿名攻击、邮件炸弹、泪滴、隐蔽监听、端口扫描、内部入侵以及涂改信息等各种各样的黑客行为无处不在，垃圾邮件等互联网污染难以防范。IP 互联网用户可以设定任意 IP 地址来冒充别人，可以向网上任何设备发出探针窥探别人的信息，也可以向网络发送任意干扰数据包。许多聪明人发明了各种防火墙试图保证安全，但是安装防火墙是自愿的，防火墙的效果是暂时的和相对的，IP 互联网本身难免被污染。这是 IP 互联网第二项安全败笔。

大一统网络用户入网后，网络交换机仅允许用户向节点服务器发送有限的服务请求，其他数据包一律拒绝。如果服务器批准用户申请，即向用户所在的交换机发出网络通行证，用户终端发出的每个数据包若不符合网络交换机端的审核条件就一律丢弃，这样就彻底杜绝了黑客攻击。每次服务结束后，自动撤销通行证。因此大一统网络不需要防火墙、杀毒、加密和内外网隔离等被动手段，从结构上彻底阻断了黑客攻击的途径，是本质上的安全网络。

网络设备与用户数据完全隔离，切断病毒扩散的生命线。IP 互联网设备可随意拆解用户数据包，大一统网络设备与用户数据完全隔离。

冯·诺依曼创造的计算机将程序指令和操作数据放在同一个地方，也就是说一段程序可以修改机器中的其他程序和数据。沿用至今的这一计算机模式，给特洛伊木马、蠕虫、病毒、和后门留下了可乘之机。随着病毒的高速积累，防毒软件和补丁永远慢一拍，处于被动状态。互联网 TCP/IP 的技术核心是尽力而为、储存转发和检错重发。为了实现互联网的使命，网络服务器和路由器必须具备解析用户数据包的能力，这同样为黑客留下了后门。网络安全从此成了比谁聪明的游戏，制作病毒与杀毒、攻击与防护，永无休止。这是 IP 互联网的第三项遗传性缺陷。

大一统网络交换机设备中的 CPU 不接触任何一个用户数据包，也就是说，整个网络只是在业务提供方和接收方的终端设备之间建立一条完全隔离和具备流量行为规范的透明管道。用户终端不管收发什么数据，一概与网络无关，从结构上切断了病毒和木马的生命线。因此大一统网络杜绝了网上的无关人员窃取用户数据的可能性，同理，那些黑客也就没有了可以攻击的对象。

用户之间的自由连接完全隔离，确保有效管理。IP 互联网是自由市场，无中间人；而大一统网络则类似百货公司，有中间人。

对于网络来说，消费者与内容提供商都属于网络用户范畴，只是大小不同而已。IP 互联网是个无管理的自由市场，任意用户之间可以直接通信。也就是说，要不要管理是用户说了算，要不要收费是单方大用户（供应商）说了算，要不要遵守法规也是单方大用户说了算。运营商至多收取入场费，要想执行法律、道德、安全和商业规矩，现在和将来都不可能。这是 IP 互联网的第四项架构上的顽疾。

大一统网络创造了服务节点的概念，形成有管理的百货公司商业模式。用户之间或者消费者和供货商之间严格禁止自由接触，一切联系都必须取得节点服务器的批准。这是实

现网络业务有效管理的必要条件。有了不可逾越的规范，才能在真正意义上实现个人与个人之间、企业与企业之间、企业与企业之间，或者统称为有管理的用户之间的对等通信。

商业规则植入通信协议，确保盈利模式。IP 互联网奉行先通信后管理的模式，大一统网络奉行先管理后通信的模式。

网上散布非法媒体内容，只有在造成恶劣影响后才能在局部范围内查封，不能防患于未然。法律与道德不能防范有组织、有计划的职业攻击，而且法律只能对已造成危害的攻击者实施处罚。IP 互联网将管理定义为一种额外附加的服务，建立在应用层。因此管理自然成为一种可有可无的摆设。这是 IP 互联网第五项难移的本性。

大一统网络用户终端只能在节点服务器许可范围内的指定业务中选择申请其中之一。服务建立过程中的协议信令由节点服务器执行。用户终端只是被动地回答服务器的提问，接受或拒绝服务，不能参加到协议建立过程中。一旦用户接受服务器提供的服务，只能按照通行证规定的方式发送数据包，任何偏离通行证规定的数据包一律在底层交换机中丢弃。大一统网络协议的基本思路是实现以服务内容为核心的商业模式，而不只是完成简单的数据交流。在这一模式下，安全成为固有的属性，而不是附加在网络上的额外服务项目。当然，业务权限审核、资源确认和计费手续等，均可轻易包含在管理合同之中。

14.4 隐私问题

随着数据挖掘技术的发展，大数据的价值越来越明显，隐私泄露问题的出现也使大家愈发重视个人隐私保护。在我国相关信息安全和隐私保护法律法规不够完善的情况下，个人信息的泄露、滥用等问题层出不穷，给人们的生活造成了很多麻烦。

14.4.1 防不胜防的隐私泄露

个人隐私的泄露，在最初阶段主要是由于黑客主动攻击造成的。人们在各种服务网站注册的账号、密码、电话、邮箱、住址、身份证号码等各种信息集中存储在各个公司的数据库中，并且同一个人在不同网站留下的信息具有一定的重叠性，这就导致一些防护能力较弱的小网站很容易被黑客攻击而造成数据流失，进而导致很多用户在一些安全防护能力较强的网站的信息也就失去了安全保障。随着移动互联网的发展，越来越多的人把信息存储在云端，越来越多的带有信息收集功能的手机 APP 被安装和使用，而当前的信息技术通过移动互联网的途径对隐私数据跟踪、收集和发布的能力已经达到了十分完善的地步，个人信息通过社交平台、移动应用、电子商务网络等途径被收集和利用，大数据分析和数据挖掘已经让越来越多的人没有了隐私。对于一个不注意个人隐私保护的人来说，网络不仅知道你的年龄、性别、职业、电话号码、爱好，甚至知道你居住的具体位置、你现在在哪里、你将要去哪里等，这绝不是危言耸听。

罗彻斯特大学的亚当·萨迪克（Adam Sadilek）和来自微软实验室的工程师约翰·克拉姆（John Krumm）收集了 32000 英里 703 个志愿者和 396 辆车的 GPS 数据并建造了一个“大规模数据集”。他们通过编写一个算法，可以大致预测一个人未来可能到达的位置，最多可以预测到 80 周后，其准确度高达 80%。

为保护个人隐私权，很多企业都会对其收集到的个人信息数据进行“匿名化”处理，抹掉能识别出具体个体的关键信息。但是在大数据时代，由于数据体量巨大，数据的关联性强，即使是经过精心加工处理的数据，也仍然可能泄露敏感的隐私信息。早在 2000 年，Latanya Sweeney 博士就表明只需要 3 个信息就可以确定 87% 的美国人：ZIP 码、出生日期和性别，而这些信息都可以在公共记录中找到。另外根据用户的搜索记录也可以很轻易地锁定某个人。美国在线（AOL）在 2006 年公布了 3 个月的将近两千万条搜索记录，虽然记录没有真实姓名，但是纽约时报的记者根据一名用户的搜索记录：“60 岁的单身男子”、“在各种东西上小便的狗”、“利尔本市的园丁”等，将该用户锁定为一位住在利尔本市的 62 岁的寡妇。

当前人们在使用社交网站发布说说、微博的同时使用定位功能显示自身准确位置，各种好友评论中无意的直呼真名或者职务，各种网站和论坛注册的邮箱、电话号码、QQ 等信息，电商平台的实名认证和银行卡关联，网上投递个人简历等都会把个人隐私信息全部或部分展示出来。同时随着移动互联网的发展，越来越多的人开始使用云存储和各种手机 APP（为了与商家合作推送广告，很多 APP 都具有获取用户位置、通讯录的功能），个人信息也就相应地在互联网和云存储中不断增多。谷歌眼镜作为互联网时代最新的科技成果之一，带给人们随时随地拍摄、随时随地上传的新鲜体验，但是这也意味着越来越多的人可能在不知情的情况下已经被录像并上传到了互联网。因此谷歌眼镜直接被冠以了“隐私杀手”的称号。这些新技术就像一把双刃剑，在方便人们生活的同时也带来了个人隐私泄露的更大风险。

14.4.2 隐私保护的政策法规

没有规矩，不成方圆。在现代社会，完善的法律法规是社会秩序正常运行的基本保障，也是各行各业健康有序发展的根本依据，互联网行业同样不能例外。当前，包括中国在内的很多国家都在完善与数据使用及隐私相关的法律，以便在保障依法合理地搜集处理和利用大数据信息创造社会价值的同时保护隐私信息不被窃取和滥用。

在隐私保护立法方面走在前面的当属欧洲。欧洲将隐私作为一种值得法律完全保护的基本人权来对待，制定了范围广泛的跨行业的法律。欧洲认为隐私是一个“数据保护”的概念，隐私是基本人权的基础，国家必须承担保护私人信息的义务。欧洲最早的数据立法是 20 世纪 70 年代初德国黑森州的数据保护法，1977 年德国颁布了《联邦数据保护法》。瑞士 1973 年通过了《数据保护法案》。1995 年 10 月，欧盟议会代表所有成员国，通过了《欧盟个人数据保护指令》，简称《欧盟隐私指令》，指令的第一条清楚地阐明了其主要目标

是“保护自然人的基本权利和自由，尤其与个人数据处理相关的隐私权”。这项指令几乎涵盖了所有处理个人数据的问题，包括个人数据处理的形式，个人数据的收集、记录、存储、修改、使用或销毁，以及网络上个人数据的收集、记录、搜寻、散布等。欧盟规定各成员国必须根据该指令调整或制定本国的个人数据保护法，以保障个人数据资料在成员国间的自由流通。1998 年 10 月，有关电子商务的《私有数据保密法》开始生效。1999 年欧盟委员会先后制定了《互联网上个人隐私权保护的一般原则》、《关于互联网上软件、硬件进行的不可见和自动化的个人数据处理的建议》、《信息公路上个人数据收集、处理过程中个人权利保护指南》等相关法规，为用户和网络服务商提供了清晰可循的隐私权保护原则，从而在成员国内有效地建立起了有关互联网隐私权保护的统一的法律体系。

作为电子商务最为发达的国家，美国在 1986 年就通过了《联邦电子通信隐私权法案》，它规定了通过截获、访问或泄露保存的通信信息侵害个人隐私权的情况、例外及责任，是处理互联网隐私权保护问题的重要法案。

与数据隐私密切相关的是数据的“所有权”和数据的“使用权”。数据由于资产化和生产要素化，其所附带的经济效益和价值也就引出了一系列法律问题，比如数据的所有权归属，其所涵盖的知识产权如何界定，如何获得数据的使用权，以及数据的衍生物如何界定等。智慧城市和大数据分析往往需要整合多种数据源进行关联分析，分析的结果能产生巨大的价值，然而这些数据源分属于不同的数据拥有者，对这些拥有者来说，数据是其核心资源甚至是保持竞争优势的根本，因此他们不一定愿意将其开放共享。如何既能保证数据拥有者的利益，又能有效促进数据的分享与整合，也将成为与立法密切相关的重要因素。

14.4.3 隐私保护技术

对于隐私保护技术效果可用“披露风险”来度量。披露风险表示攻击者根据所发布的数据和其他相关的背景知识，能够披露隐私的概率。那么隐私保护的目的就是尽可能降低披露风险。隐私保护技术大致可以分为以下几类。

1. 基于数据失真（Distortion）的技术

数据失真技术简单来说就是对原始数据“掺沙子”，让敏感的数据不容易被识别出来，但沙子也不能掺得太多，否则就会改变数据的性质。攻击者通过发布的失真数据不能还原出真实的原始数据，但同时失真后的数据仍然保持某些性质不变。比如对原始数据加入随机噪声，可以实现对真实数据的隐藏。当前，基于数据失真的隐私保护技术包括随机化、阻塞（Blocking）、交换、凝聚（Condensation）等。例如，随机化中的随机扰动技术可以在不暴露原始数据的情况下进行多种数据挖掘操作。由于通过扰动数据重构后的数据分布几乎等同于原始数据的分布，因此利用重构数据的分布进行决策树分类器训练后，得到的决策树能很好地对数据进行分类。而在关联规则挖掘中，可以在原始数据中加入很多虚假的购物信息，以保护用户的购物隐私，但同时又不影响最终的关联分析结果。

2. 基于数据加密的技术

在分布式环境下实现隐私保护要解决的首要问题是通信的安全性，而加密技术正好满足了这一需求，因此基于数据加密的隐私保护技术多用于分布式应用中，如分布式数据挖掘、分布式安全查询、几何计算、科学计算等。在分布式环境下，具体应用通常会依赖于数据的存储模式和站点（Site）的可信度及其行为。

对数据加密可以起到有效地保护数据的作用，但就像把东西锁在箱子里，别人拿不到，自己要用也很不方便。如果在加密的同时还想从加密之后的数据中获取有效的信息，应该怎么办？最近在“隐私同态”或“同态加密”领域取得的突破可以解决这一问题。同态加密是一种加密形式，它允许人们对密文进行特定的代数运算，得到的仍然是加密的结果，与对明文进行运算后加密一样。这项技术使得人们可以在加密的数据中进行诸如检索、比较等操作，得出正确的结果，而在整个处理过程中无须对数据进行解密。比如，医疗机构可以把病人的医疗记录数据加密后发给计算服务提供商，服务商不用对数据解密就可以对数据进行处理，处理完的结果仍以加密形式发送给客户，客户在自己的系统上才能进行解密，看到真实的结果。但目前这种技术还处在初始阶段，所支持的计算方式非常有限，同时处理的时间开销也比较大。

3. 基于限制发布的技术

限制发布也就是有选择地发布原始数据、不发布或发布精度较低的敏感数据，实现隐私保护。这类技术的研究主要集中于“数据匿名化”，就是在隐私披露风险和数据精度间进行折中，有选择地发布敏感数据或可能披露敏感数据的信息，但保证对敏感数据及隐私的披露风险在可容忍范围内。数据匿名化研究主要集中在两个方面：一是研究设计更好的匿名化原则，使遵循此原则发布的数据既能很好地保护隐私，又具有较大的利用价值；二是针对特定匿名化原则设计更“高效”的匿名化算法。数据匿名化一般采用两种基本操作：一是抑制，抑制某数据项，亦即不发布该数据项，比如隐私数据中有的可以显性标识一个人的姓名、身份证号等信息；二是泛化，泛化是对数据进行更概括、抽象的描述。譬如，将年龄3泛化为[0, 5]，把详细住址泛化为某个城区或乡镇等，可以降低信息的精确性，起到一定的隐私保护作用。

安全和隐私是云计算和大数据等新一代信息技术发挥其核心优势的拦路虎，是大数据时代面临的一个严峻挑战。但是这同时也是一个机遇，在安全与隐私的挑战下，信息安全和网络安全技术也得到了快速发展，未来安全即服务（Security as a Service）将借助云的强大能力，成为保护数据和隐私的一大利器，更多的个人和企业将从中受益。历史的经验和辩证唯物主义的原理告诉我们，事物总是按照其内在规律向前发展的，对立的矛盾往往会在更高的层次上达成统一，矛盾的化解也就意味着发展的更进一步。相信随着相关法律体系的完善和技术的发展，未来大数据和云计算中的安全隐私问题将会得到妥善解决。

14.5 练习题

1. 传统信息安全防护主要包括哪几个方面?
2. 从数据保护的角度看, 云安全的防护手段有哪些?
3. 如何创立本质安全的新型 IT 体系?
4. 简述隐私保护技术分类。

参考文献

- [1] Brodtkin J. GartnerSeven: Cloud-computing Security Risks. Info World Security Central News[EB/OL]. [http://www.idi.ntnu.no/emner/tdt60/papers/Cloud Computing Security Risk.pdf](http://www.idi.ntnu.no/emner/tdt60/papers/Cloud%20Computing%20Security%20Risk.pdf)
- [2] 冯登国, 张敏, 张妍, 徐震. 云计算安全研究. Journal of Software, 2011, 22(1):71-83.
- [3] Crampton J, Martin K, Wild P. On key assignment for hierarchical access control. In: Guttan J, ed. Proc. of the 19th IEEE Computer Security Foundations Workshop—CSFW 2006. Venice: IEEE Computer Society Press, 2006, 5-7.
- [4] Goyal V, Pandey A, Sahai A, Waters B. Attribute-Based encryption for fine-grained access control of encrypted data. In: Juels A, Wright RN, Vimecati SDC, eds. Proc. of the 13th ACM Conf. on Computer and Communications Security, CCS 2006. Alexandria: ACM Press, 2006, 89-98.
- [5] Bethencourt J, Sahai A, Waters B. Ciphertext-Policy attribute-based encryption. In: Shands D, ed. Proc. of the 2007 IEEE Symp. on Security and Privacy. Oakland: IEEE Computer Society, 2007, 321-334.
- [6] Chang YC, Mitzenmacher M. Privacy preserving keyword searches on remote encrypted data. In: Ioannidis J, Keromytis AD, Yung M, eds. LNCS 3531. New York: Springer-Verlag, 2005, 442-455.
- [7] Goh EJ. Secure indexes. Technical Report, Stanford University, <http://eprint.iacr.org/2003/216/>

第15章

大数据技术发展趋势

在过去的研究和实践中,许多研究学者从体系结构、编程模型、算法设计、并行处理、存储机制等多方面都提出了很多对 IT 发展有很大帮助的意见。但是这些体系、方法和策略在大数据技术日益发展的今天,却不能够解决大数据所提出的难题。大数据技术的发展和应用的多样化,对 IT 技术架构和数据处理技术等提出了新的挑战。大数据的增长速度快、时间局部性低等特点,使得以计算为中心的传统模式面临着内存容量有限、I/O 压力大、难以取得较好的性能等问题。大数据是一种以数据为中心的数据密集型技术,现有的以计算为中心的技术难以满足大数据的应用需求,因此,整个 IT 架构的革命性重构势在必行。这就为内存计算、实时计算以及泛在计算等技术的发展带来了机遇。

15.1 实时化

随着大数据的爆发和迅速增长,对海量数据的即时处理要求也越来越迫切。实时计算的应用场景也越来越多。实时计算要求对用户的响应接近零延时,给人们提供的内容都是最新的。未来,互联网与物联网、虚拟现实和流媒体技术的结合都需要实时计算。

在实时计算时代,企业应该大力改进自己的实时搜索、实时预测、实时服务等实时计算能力。电子商务网站应能够实时处理并挖掘用户行为产生的数据,能够对用户行为进行预测,并实时为用户推荐可能感兴趣的商品和广告;新闻类网站需要保证新闻的时效性,将最新的新闻推送给用户;社交网站可以将一个实时热点进行聚合,将用户聚集为一个圈子。在实时预测方面还包括实时预测股价变化、商品价格变化、人流变化、路况变化、交

通情况和更短时间内的天气预测等，这些预测可以帮助人们提早做出决策。在实时服务方面，商家应能够与用户实时沟通，使用户获得更好的体验。这些业务对时效性的高要求，也对实时计算技术提出了新的挑战。

目前已经有较多的实时计算框架，例如 Spark，它是一个实时计算系统，可支持流式计算、批处理和实时查询。除了 Spark，还有 Yahoo 的 S4、Twitter 的 Storm、IBM 的 StreamBase 等。

Spark 从 2009 年诞生，2010 年开源，2013 年成为 Apache 的孵化项目，到 2014 年成为 Apache 的顶级项目，发展过程不过五年的时间。与 Hadoop 相比，Spark 在性能和方案的统一性方面都具有较大的优势。它提供基于 RDD 的一体化解决方案，将 MapReduce、流计算、SQL、机器学习、图计算等模型进行统一，以一致的 API 公开，并提供相同的部署方案，使得 Spark 工程的应用领域相当广泛。同时，Spark 还提供了对 Java、Scala、Python 和 R 语言的支持。在 Spark 的当前版本中，在机器学习方面已经支持了超过 15 种算法，包括决策树、PCA、SVD、LBFGS 等。

在大数据领域，目前最火的应该就是 Spark，Spark 的发展速度也极其惊人。以 Spark 的发展路线，在未来将会发布的版本中，Spark 的新方向是数据科学与平台化。Spark 1.3 正式发布了 DataFrame，在 1.4 中会将 Spark 和 R 相结合，推出 Spark R。另外，Spark 也会基于 DataSource 接口无缝接入各个不同的数据源，这不仅给不同数据源的使用者提供了更便利的 Spark 使用方式，更给那些需要从不同数据源收集数据，并结合起来进行分析挖掘的用户提供了一个极其简单的实现。目前，Spark 已经得到了业界的广泛认可，在 IBM、Cloudera、Hortonworks 等企业的 Hadoop 版本中都已经包含了 Spark。Spark 也已经被许多互联网企业应用在商业项目中，国内的百度、阿里、腾讯、网易、搜狐等，都已经投身 Spark 的阵营。Spark 还被应用在音乐推荐、文本分析、客户智能实时推荐、实时审计的数据分析等多个方面。

除了 Storm 和 Spark Streaming 以外，还有其他一些实时处理框架，具体如下。

Impala: Google Dremel 的开源实现（与 Apache Drill 类似）。因为交互式实时计算需求，Cloudera 推出了 Impala 系统，该系统适用于交互式实时处理场景，要求最后产生的数据量一定要少。

StreamBase: 由 IBM 开发的一个商业流式计算系统，应用于金融行业和政府部门。使用 Java 开发，它提供了较多的算子、功能和其他组件来帮助构建应用程序，并且提供了类 SQL 语言来描述计算过程。

Stinger (Tez optimized Hive): 由 Hortonworks 开源，可以理解为 Google Pregel 的开源实现。它可以像 MapReduce 一样，用来设计 DAG 应用程序，但 Tez 只能运行在 YARN 上。Tez 的一个重要应用是优化 Hive 和 PIG 这种典型的 DAG 应用场景，它通过减少数据读写 IO，优化 DAG 流程，使 Hive 速度提高了很多倍。

Presto: 它是一个分布式的 SQL 查询引擎，于 2013 年 11 月由 Facebook 开源。它专门用来进行高速、实时的数据分析。支持标准的 ANSI SQL，包括复杂查询、聚合（aggregation）、

连接 (join) 和窗口函数 (window functions)。Presto 设计了一个简单的数据存储的抽象层, 以满足在不同数据存储系统 (包括 HBase、HDFS、Scribe 等) 之上都可以使用 SQL 进行查询。

大数据技术的发展, 让实时分布式计算系统开始占有举足轻重的地位。用户对于数据处理系统实时性的需求必将推动实时计算技术的快速发展。未来在实时计算系统方面必然会出现更多产品和应用。

15.2 内存计算

由于目前廉价计算机的运行环境和批处理高度优化的系统结构的限制, 现有的数据管理技术和解决方案不能很好地适应新型实时、交互式的复杂业务需求, 而更适用于对 Web 数据的处理。大数据具有增长速度快、数据规模大、数据类型多样等特征, 也对当前的计算模式提出了新的挑战。现有的计算模式在面对大数据处理时存在着内存容量有限、I/O 效率低下、并发控制困难、数据处理总体性能较低等许多问题。因此, 基于内存的数据管理技术应运而生。

内存计算是以大数据为中心, 通过对体系结构及编程模型等进行重大的革新, 最终显著提升数据处理能力的新型计算模式。

15.2.1 机遇与挑战

高效的数据管理技术一直伴随着 IT 技术的发展。而在大数据的环境下, 用数据说话已经成为数字化社会的突出特色, 也就对数据的存储计算能力提出了新的要求。数据管理技术面临着新的、大量的机遇和挑战。

在大数据时代, 企业竞争日趋激烈, 如果企业不能对用户的需求实时做出反应, 将会损失大量的用户。因此, 应用对时效性的需求为内存计算提供了发展的内在驱动力。在大数据环境下, 对于商业价值的挖掘方式在逐渐发生变化。企业的需求更多地表现在实时动态计算、交互式分析、即时查询等新的实时业务方面。以风险管理为例, 如果企业不能实时地识别出交易中的欺诈行为, 或者在交易过程中进行实时预警, 而只提供有些事后的补救措施, 将会给企业和用户带来不可估计的损失。不仅是传统应用, 一些新兴的业务和行业的出现, 也急剧增多了对时效性分析的需求量, 如电商高频交易分析、社交网络的群体性事件预警、智能电网用电信息采集、基于位置的智能推荐与计算广告等。内存计算能够将企业处理信息的质量和速度都提高到前所未有的水平, 可以帮助企业实现实时业务信息的快速提取, 不仅能够为企业带来独特的商机, 而且是企业在未来激烈的市场竞争中取胜的关键所在。

除了对实时性业务处理的需求外, 新型硬件技术的发展也为内存计算的发展提供了强

大的支撑。首先,从 2005 年开始,多/众核技术、异构多核集成技术和多 CPU 的并行处理技术的发展,为提高处理能力和运算速度提供了新的途径。其次,多核共享缓存以及多处理器共享内存的新型架构的出现,进一步促进了非一致性内存访问(NUMA)技术的出现与发展。最后,近年来也出现了一些新的存储技术开始产品化,例如闪存、相变存储器(PCM)、磁阴式随机存储器(MRAM)和电阴式随机存储器(RRAM)等,它们被称为存储级内存(SCM),具有非易失性、随机访问延迟小、并行度高、功耗低、片载密度高等优良特性。然而如何有效地将 SCM 部署在系统上还存在着较大研究空间。

近年来,计算机硬件迅猛发展,数据处理环境和数据特征已经发生变化,这也为内存计算提供了良好的发展机会。主要表现在以下几个方面。

① 要处理的数据无法长期存储在内存的场景将发生变化。随着 SCM 技术的发展,内存容量越来越大,价格也越来越便宜,适于用内存计算的拥有 TB 级内存容量的服务器正在逐渐普及。

② 数据处理系统最主要的性能瓶颈由磁盘 I/O 逐渐向网络 and 内存间 I/O 转移。

③ 磁盘数据访问的局部性将不再是性能优化的主要目标,而由内存数据访问的局部性代替。与磁盘数据访问局部性相比,内存在层次结构、缓存容量、对齐模式、缓存介绍等多个方面都存在着较大的差异。

另外,数据处理模式从 SQL 到 NoSQL 再到 NewSQL 的变迁,也为内存计算提供了发展方向。SQL 虽然在关系数据处理模式上较为成功,但在可处理数据类型多样性、实时性和性价比等方面存在着较大的瓶颈。而 NoSQL 虽然提供了良好的可扩展性,却在一致性保证方面受到了一定的限制。在保证可扩展性、高性能的前提下,支持数据库事务的 ACID 特性成为数据库技术新的发展方向。基于内存计算的 NewSQL 就成为了新的可选的数据处理模式。

15.2.2 研究进展

近年来,在大数据浪潮的推动下,分布式内存计算已经取得了较多的进展,并且受到越来越多的关注。下面从并行编程模型、混合存储体系结构和内存数据管理三个方面来介绍目前已经取得的研究成果。

1. 并行编程模型

分布式并行编程模型架构在硬件和应用之间,它包括了存储模型、执行模型、调度模型。它可以在大规模廉价集群中以并行、可扩展、容错、易用、透明的方式支持各种应用的有效执行。

目前,面向批处理的编程模型有 Google 的 MapReduce 和微软的 Dryad 等。该类编程模型在可扩展性、易用性和性价比方面都有比较大的优势。它们虽然可以较好地处理批量的数据,但对流式数据的实时性处理在性能上就难以满足。以 MapReduce 为例,导致它处

理数据时高延时的原因有很多,例如依赖于磁盘的容错机制、Map 和 Reduce 阶段之间阻塞流水线的障碍、基于 pull model 的数据传输方式等。

基于大数据处理实时性较高的要求,较多的学者对 MapReduce 系统进行了实时性优化。例如, HOP 在 Map 和 Reduce 之间建立了流水线式的传输渠道,从而消除了导致阻塞的任务间障碍,但却依然采用基于磁盘的中间数据缓存策略,因而具有较高的 I/O 开销。虽然有较多的学者在 MapReduce 优化上做了较多的工作,但却都存在各种各样的问题,与满足 BI 应用的实时性需求还存在较大的差距。

基于实时应用的需求, MapReduce 批处理的模式将会被打破,新型的面向内存的编程模型及系统都在不断涌现。最具代表性的是基于内存的分布式并行处理框架 Spark。Spark 利用内存计算有效地保证了处理的实时性要求,同时提供了交互式的迭代分析能力。流式应用也是实时性较高的一类应用系统,具有代表性的有 Twitter 的 Storm、雅虎的 S4、Facebook 的 Puma、谷歌的 MillWheel 等。Spark 也有自己的流处理框架 Spark Streaming。这些框架都可以与企业自身的具体需求相结合,可以解决一些实际的应用问题。

2. 混合存储体系结构

大规模并行计算系统的计算能力与存储子系统访问性能间存在着较大的差距,存储子系统是大数据计算系统性能的主要瓶颈。而 SCM 正在逐渐缩小持久性存储和易失性内存之间的差距。与磁盘的机械特性相比, SCM 的电气特性具有自己的独特性,因此以往适用的技术或方法已经不再适用。目前已经有较多的研究学者在 SCM 的索引技术、查询优化、容错机制、缓存与替换策略、上层算法等多方面进行了研究,对其特性进行优化,但仍然有较大的研究空间,这也必然是将来的一个研究方向和发展趋势。而面向混合存储体系的优化也具有更大的实用性和适用性。

3. 内存数据管理

传统的关系型数据库主要面向的是磁盘的存储环境,难以应用在内存环境中。同时,面向磁盘的设计和优化的数据管理系统在全内存的工作环境中也难以获得预期的性能提升。因此,就催生出了重新设计的基于高性能分布式集群的内存数据库管理系统。

近年来,工业界已经出现了一些基于内存的分布式数据库的相关产品,如最著名的全内存式数据存取系统 Memcached,已经被 Facebook、Twitter、YouTube 等多家知名企业应用。与 Memcached 类似的还有性能卓越的内存存储系统 Redis,它提供更加灵活的缓存失效策略和持久化机制。另外,还有 SAP 的 HANA、微软的 Hekaton 等内存数据库产品也在不断涌现。

除了工业界,在学术界也有一些内存数据管理系统出现。例如 MIT 的 H-Store,其根据 CPU Core 进行数据分区,通过数据库多副本来获得数据的持久性。

基于内存的数据处理技术也会对索引、查询等数据库操作带来影响。基于磁盘的 I/O 索引、查询和优化都很难直接应用于新型存储环境中,因此,这也是一个全新的研究方向,目前已有研究学者在这方面开展研究。

15.2.3 发展展望

大数据时代的来临,让内存计算在不同行业中的应用不再只是一个愿望,而正在慢慢变成现实。大数据也为企业数据的应用和处理带来空前的机遇和挑战。对于大多数企业而言,对数据的处理速度,才是真正阻碍企业高效使用数据的关键性因素。内存计算的产生,为企业的发展带来了机会。内存数据管理技术的发展主要表现在以下几个方面。

1. 与企业的应用现状相结合

目前内存计算的发展尚不成熟,企业仍会对内存计算技术的可靠性、可扩展性、安全性和部署成本等问题存在较多的顾虑。但事实证明,内存计算已经被成功应用在企业的业务系统中,并且被证明有效。内存计算技术正以迅猛的劲头逐渐占领市场。

目前内存计算技术可用的行业越来越多,已经涵盖零售、电信、金融、医疗、制造、交通、公共事业等各行各业。内存计算可以帮助企业实现在客户服务分析、营销分析、供应链和运营分析、财务分析、盈利分析、网站点击流量分析等众多功能上的实时响应能力。并且,内存计算技术将会在更多行业的更多方面实现更大规模的应用。

2. 充分利用已有技术资源

充分利用已有 SQL 和 NoSQL 技术,将二者的优势相结合,在保证数据管理系统高可扩展性的同时,提供传统的事务的 ACID 保证,从而扩大已有技术的使用范围。NewSQL 作为完全重新架构在内存计算环境下的数据管理技术,也成为新的重要发展方向。

3. 加强科研管理,推动多方参与

内存计算技术作为大数据技术链上的最新技术,近年来才逐渐为人们所关注,在发展过程中不仅需要政府部门在政策上给予支持,还需要科技部门进行积极的引导和推动。内存计算技术可以使数据分析更及时,预测结果更准确,能够为企业提供更的利益空间。在国内,农夫山泉是最典型的成功案例,通过上线 SAP HANA,该公司的报表展现速度提高了 200~300 倍,运费报表的展现从以前的 24 小时提速到 37 秒,大幅提升了企业应对市场变化的响应能力。因此,企业也具有参与内存计算技术应用与研发的内在驱动力。

15.3 泛在化

泛在计算,也可称为普适计算或环境智能。泛在计算强调和环境融为一体,计算机本身从人们视线中消失。在泛在计算环境中,人们能够在任何时间、任何地点、以任何方式进行信息的获取与处理,而这个过程是在计算设备的帮助下高度自动化完成的。比如身上的眼镜、手表、手机、鞋子等都可以进行计算。它的核心思想是将小型、便宜、网络化的

设备广泛分布在日常生活中的各个场所, 计算设备不再只是依靠命令行、图形界面等, 而是以更“自然”的方式进行交互。泛在计算的目的是要建立一个充满计算和通信能力的环境, 并使这个环境和人们逐渐融合在一起, 使人们可以随时随地透明地获得数字化服务。

泛在计算最重要的应用方向是各类信息终端产品。随着信息需求服务的多样化, 信息终端产品的形式越来越多, 功能也越来越完善, 在一个小设备中往往能集成多种功能。智能空间也是泛在计算的重要应用, 例如智能会议室、智能教室、作战指挥室等。泛在计算还可以应用于商业识别 RFID 芯片。RFID 芯片可以代替商品上的条件形码, 在提供商品的名称、价格之外, 还可以集成其他更有用的信息, 例如衣服的原料、产地、规格, 药品的成分、适用症、禁忌等。

目前, 各类新型技术的发展, 已经为泛在计算的发展提供了较新的平台。云计算技术的发展几近无限地扩展了终端的存储和计算能力。物联网技术的发展也重新定义了物物之间的关系, 加快了物体之间的通信和信息共享。智能终端和移动互联网的爆炸式增长, 让用户和终端紧密结合, 基于用户需求的服务迅猛发展。智能手机、智能电视、智能汽车等都在逐渐普及, 即感知和计算能力, 也泛在地嵌入物理环境中。

15.3.1 发展现状

在国外, 智能手机、可穿戴设备、智能汽车都已经取得了较大进展。智能手机的存储和计算能力早已超过早期的 PC, 同时智能手机的功能也在被云计算无限扩展。而随着传感技术的进步, 微型可穿戴设备也在不断涌现, 它们可通过与智能手机相连接, 提供更好的人机交互界面、数据存储和云服务访问。例如, MYO 臂带可通过检测手势变化时的肌肉放电来识别用户手势。同时, 由于可穿戴设备可以隐藏于衣服、鞋子等日常生活用品中, 因此可实现时时刻刻感知人的活动, 例如耐克的篮球鞋通过内置芯片可以感知用户运动时跳起的高度和跑步的速度等。而智能汽车则更关注汽车本身的智能化, 帮助司机完成更加完全和可靠的驾驶。该领域的两个研究热点是车联网和自动驾驶。例如, 通用汽车在 2010 年 3 月发布了基于车联网的概念车, 可通过城市智能交通网络和车辆自身驾驶系统实现智能导航和行驶。谷歌公司的无人驾驶车也已累计行驶各种路况 30 万英里。

在人机交互方面, 泛在计算充分融合了传感器技术、智能计算技术和嵌入式系统等领域的最新技术, 将交互设备隐藏起来, 朝着以人为本、方便用户的方向发展。交互设备的隐藏表现在多尺度显示屏幕和可穿戴设备的普及两个方面。多尺度的显示屏幕分别以智能手机、智能电视和大型投影屏为代表, 为多种交互手段提供了可能; 而将芯片植入手表、鞋子、眼镜等可穿戴设备, 可以提供无缝感知和交互体验。同时, 交互的广度和深度也在不断发展。感知技术的进步使得多种交互手段的融合成为可能, 交互的方式不断创新, 而复杂信号感知技术使人体交互的层面不断深入。

在国内, 智能终端已经广泛普及, 通信网络也大量铺设, 在网络运营、服务提供、终端开发等各个方面都出现了一批领头羊, 积累了较多关键技术和产品, 正在逐步形成非常

有利于泛在计算发展和应用的生态环境。与国外相比,我国的泛在计算研究在基础设施、用户服务、交互理论等方面都表现不俗,部分成果也受到国内外研究者的好评和引用。但是,国内的原始创新能力仍有待加强。在硬件方面,感知设备、可穿戴设备等基础硬件大都源自国外,这体现了我国在研发制造上的弱点,也体现出学术机构和生产企业对接的重要性。

另外,在游戏行业,一直存在着这样一种观点,即随着游戏技术的发展和玩家需求扩大,未来游戏终将走向沉浸式体验。玩家通过一些外部设备来实现虚拟现实和增强现实的体验,仿佛置身在真实游戏的世界中,并且获得相应的身体感知。目前已经有较多的科技企业推出与虚拟现实技术相关的研发成果,如微软的 SurroundWeb、索尼的 Project Morpheus、谷歌的 Cardboard 和三星的 Gear VR。

而在国内,也已经有人提供虚拟现实的解决方案。例如,Nibiru 游戏平台在 2014 年的中国互联网大会上与香港维爱科技有限公司共同发布了虚拟现实解决方案。该方案以智能手机为播放平台,以 VRTRID 3D 头盔式眼镜为虚拟现实载体,融合了 Nibiru 平台的多款高清 3D 游戏解决方案,为玩家带来真正的沉浸式游戏体验。Nibiru VRTRID 3D 虚拟现实眼镜,还可以将手机播放的 3D 视频转化为梦幻般的虚拟现实效果。同时,暴风影音在国内率先正式开通了 3D 视频服务,为玩家提供了丰富的内容源;而中国 4G 手机领航者酷派,也完全融合了 Nibiru VRTRID 3D 虚拟现实解决方案。

总之,目前在国内外,泛在计算已经取得了较多的研究成果,人们也因此受益良多。正在迅速发展的大数据技术为泛在计算的发展提供了良好的机会,与现有基础设施、软硬件环境、计算能力、新材料等相结合,打造以人为本的泛在计算产品和服务,是目前发展的重要趋势。

15.3.2 发展趋势

泛在计算和云计算、大数据的发展,在未来将会为人们的生活带来较大的改变。并且,围绕人的计算也将是泛在计算在未来的一个最明显的特征。

例如,从智能家庭到智慧城市,未来人们的居住环境将由一个个具有智能的家庭单位连接成网络,实现资源的共享和优化配置。通过各种传感器设备的大量应用,智能家庭单位中的功能单元将能理解用户的行为习惯和生活规律,并通过与用户社交网络的融合,主动与其进行人性化互动。用户也可以直接通过语音、表情、手势等与家居设备进行交互。

人们将通过车联网实现智能出行。智能出行会将目前建立的城市交通信息网络、智能电网以及社区信息网络连接起来,提供快速、安全、环保的出行体验。未来的汽车将具备高度智能的车载信息系统,可以随时随地获得即时资讯,并且及时做出与交通有关的决定。同时,未来的路网将设计成立体式交通网络,允许不同速度和模式的车辆快速通行。

未来的移动社交网络会逐步将在线交互与位置和社会活动等线下感知信息相结合,而不再是完全虚拟的网络。人们可以通过移动社交网络平台成为朋友,增加线上线下的交流。

社会感知是泛在计算的高级阶段。群智感知是实现社会感知的有效途径。在未来的群智感知中，每个人都会充当多个角色。

在上述应用的驱动下，泛在计算将在大数据的推动下，在未来实现较大的发展。其在技术上也会面临较多的挑战。目前，由于物联网、云计算、大数据、社交网络等技术的发展，使人与设备的互动性增强，传统的泛在计算模型已经不再适用于当前的应用环境，需要及时做出改变；信息空间、物理空间和社会空间逐渐融合，信息已经被深度嵌入物理环境中，因此，迫切需要建立新型的信息基础设施为构建泛在计算环境服务。而随着大量异构传感设备的使用，大规模、异构、多类型的数据挖掘和处理也将成为泛在计算的一个重大挑战，包括低质量数据的过滤、大数据的理解和处理技术等。

而随着云计算、社交网络、物联网及其他基础设施的发展，以及各种异构终端的兴起，形成了以人为中心的普适感知源和服务源。以可穿戴设备为典型代表的一批微型终端，通过与智能手机相连接，可以极大地扩展现有设备的感知和交互能力，并能记录关于人的各个维度的信息，从而提供个性化的服务。

泛在计算在人机交互领域正处于茁壮成长时期，充分融合了智能计算技术、传感器技术以及嵌入式系统等领域的最新技术，在深度和广度上均达到了前所未有的水平。每一种新的生理信号均可能延伸为一种新型人机交互通道，利用这些新型生理信号的基于手势、脑电等新型感知方式的人机交互模式日渐兴起，将为人们提供更为直观自然的交互体验。

泛在计算与社交网络深度融合带来线上线下行为的相互映照，通过线上数据可以对物理事件进行预测，而通过物理世界交互记录可以进一步增强线上交流。未来的挑战包括如何基于线上线下特征对用户进行建模，如何通过线上线下特征的融合进行行为预测等。

1. 可穿戴设备

可穿戴设备是可以直接穿在身上，或者整合到用户的衣服或配件中的一种便携式设备。它不仅仅是一种硬件设备，它可以通过软件支持、数据交互和云端交互来实现强大的功能。可穿戴设备将会为我们的生活、感知带来很大的改变。

英特尔公司已与迈克尔·J·福克斯帕金森研究基金会合作，利用可穿戴设备来监测帕金森病人的症状，并收集和分析相关数据，从中找出规律性的东西。该公司称，他们现在的临床工作旨在改善帕金森病的研究和治疗，并将率先推出新的大数据分析平台来监测帕金森病人的病情发展。这样一来，研究者将可以针对帕金森病人研究出更好的药物和治疗方法。

而 Veristride 公司开发出的“大数据分析”方法，可以给正在康复中的中风患者、截肢者或其他病人及其临床医生提供实时信息。Veristride 公司的技术已开始在截肢者和中风患者身上进行测试。他们还综合利用智能鞋垫、智能手机应用程序和数据分析平台来给病人及其治疗专家提供有用的信息，包括病人过去和现在的病况，以及在此基础上分析预测的病人将来可能会出现的情况。Veristride 公司的解决方案并不是简单地记录和汇报信息，它还会识别病人步态的异常，提供适合病人的活动指南，跟踪病人的病情改变，以及根据现

有情况提供预测信息。显而易见，一天提供足足 8 个小时的实时信息，这将有助于有效地监测病人的病情及其康复情况，并提供积极有益的指导。

在这两个例子中，大数据分析是整个解决方案的灵魂，而可穿戴传感器只是收集信息的窗口。由于这些数据均被用于解决棘手的问题，因此这两种方案均具有实际的市场前景。

2. 可植入设备

(1) 可植入式智能手机

现在，人们几乎已经实现与手机 24 小时虚拟连接，那么与手机实现物理连接会怎样呢？2013 年，艺术家安东尼·安东尼利斯（Anthony Antonellis）将 RFID 晶片植入自己的手臂中，成为“数码纹身”，用以储存和向智能手机中转移艺术品图片。研究人员也在试验嵌入式传感器，将人体骨骼变成活体传声器。其他科学家正研究眼睛嵌入设备，可以通过眨眼捕捉图像，然后将其发送到任何本地存储设备中。如果将智能手机植入体内，显示屏应出现在哪里？知名跨国软件企业 Autodesk 的技术人员正测试一种系统，它可以通过人造皮肤展示图像。

(2) 治愈芯片

现在，患者可以使用与智能手机应用直接相连的网络植入设备监测和治疗疾病。波士顿大学正测试一种新的仿生胰腺，可植入式针头上附有微型传感器，它可与监测糖尿病患者血糖水平的智能手机应用直接对话。伦敦科学家正开发一种胶囊大小的电路，它可监测肥胖病人的脂肪水平，生成让他们感觉吃饱的遗传物质。这种电路有望取代手术或其他减肥方法。还有数十个团队正研究可监控心脏状况的植入设备。

(3) 可与医生对话的网络药丸

可植入设备不仅可与手机交流，也能与医生对话。在名为 Proteus 的项目中，英国科学家正研发一种网络药丸，它有微处理器，可以在人体内部直接给医生发送短信。这些药丸可分享病人的体内信息，帮助医生了解病人的健康状况，以及服药是否有预期效果等。

(4) 智能纹身

纹身通常被视为时髦的象征，那为何不植入一个智能纹身呢？智能纹身不仅看起来很酷，而且很有用，可解锁汽车或智能手机。美国伊利诺伊大学的研究人员已经研发出一种可植入的计算机纤维皮肤网格，它比人类头发更细，可以从里到外监控人们的身体状况。一家名为 Dangerous Things 的公司开发的 NFC 芯片，可以像纹身那样植入手指，人们可以通过它解锁。得克萨斯州的一个研究团队研究出一种可以注射至皮下的微粒，可追踪人体代谢过程。

(5) 脑机界面

人类大脑与电脑直接相连曾是科幻小说中的幻想，现在布朗大学 BrainGate 团队正研究大脑与电脑在现实中对接。他们在网站上写道：“将婴儿版阿司匹林大小的电极植入大脑中，初步研究显示神经信号可被电脑实时解码，并用于操控外部设备。”芯片制造商英特尔预测，到 2020 年脑机界面将投入实际应用。英特尔的科学家迪恩·波默洛（Dean Pomerleau）近

来撰文称：“最终，人类将更愿意向大脑中植入设备，你可以通过思想的力量进行网上冲浪。”

（6）可溶性生物电池

对于可植入设备来说，一个重大挑战是如何为这些植入体内的设备提供能源。你无法将这些能源塞入体内，将设备取出来替换电池也不容易。美国马萨诸塞州 Draper Laboratory 的科学家们正研发一种可生物降解的电池。它可在体内发电，并将其无线传输到需要的地方，然后消融。其他研究包括利用人体内的葡萄糖为可植入设备提供动力。比如马铃薯电池，尽管体积很小，但却更为先进（图 15.1）。

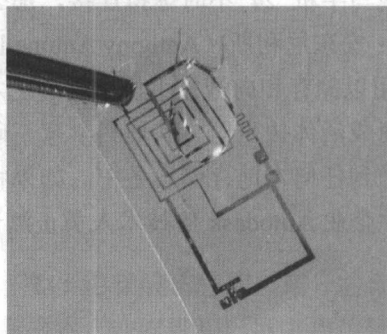


图 15.1 可溶性生物电池

（7）智能尘埃

当前最令人吃惊的可植入设备当属智能尘埃，这是一种有天线的微电脑阵列，每个都比沙粒更小，它们能在人体内自我组合成为需要的网络，处理人体内复杂情况（图 15.2）。这些纳米器件可攻击早期癌细胞，减轻伤口疼痛，甚至以加密方式存储关键个人信息等。有了智能尘埃，医生将可以在人体内进行手术，而无须开刀。信息被存储在人体内，形成个人的纳米网络，只有自己能解密。

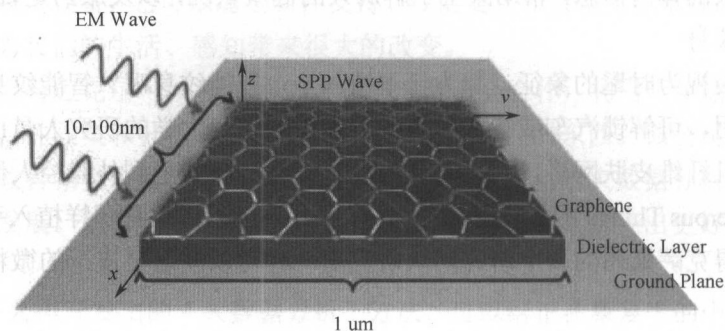


图 15.2 智能尘埃

3. 沉浸式虚拟现实

沉浸式虚拟现实（Immersive VR）提供参与者完全沉浸的体验，使用户有一种置身于虚拟世界之中的感觉。其明显的特点是：利用头盔显示器把用户的视觉、听觉封闭起来，

产生虚拟视觉；同时，它利用数据手套把用户的手感通道封闭起来，产生虚拟触动感。系统采用语音识别器让参与者对系统主机下达操作命令，与此同时，头、手、眼均有相应的头部跟踪器、手部跟踪器、眼睛视向跟踪器的追踪，使系统达到尽可能高的实时性。常见的沉浸式系统有基于头盔式显示器的系统、投影式虚拟现实系统。

一项名为“沉浸式显示体验”的专利最近被美国专利局正式对外公布，它是由微软公司在 2011 年年初申请的。据专利文件描述，一个标准视频游戏系统在连接到“环境显示器”后，能够投射出“几乎围绕用户”的全景图像（图 15.3）。

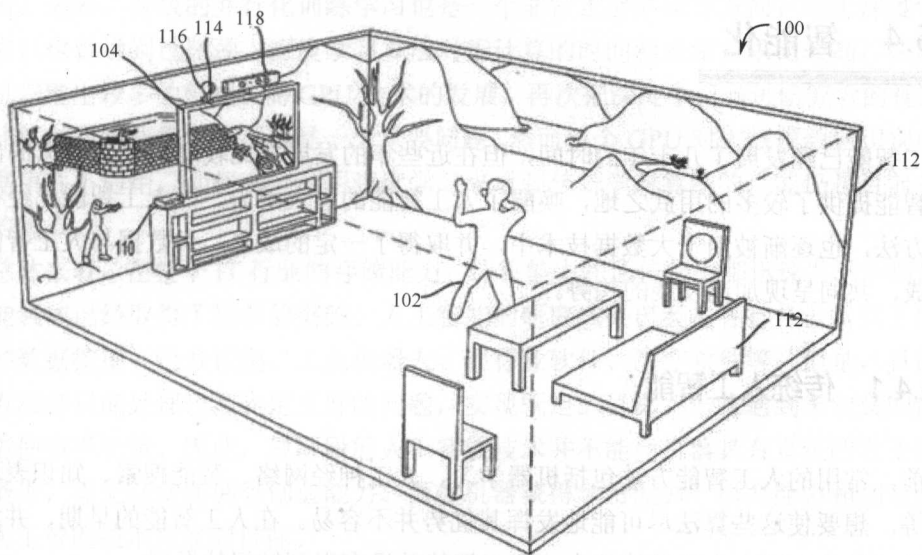


图 15.3 沉浸式显示体验

微软可以利用该技术让主机游戏的显示效果突破电视屏幕的尺寸限制，在房间内投射出一个“外围图像”，为用户构建出能够 360° 观看的虚拟场景。

与这套显示设备紧密配合工作的还有景深传感摄像系统，它可能采用 Kinect 那样的光学结构传感摄像头，也可能是更复杂的模型。它主要用以完成多图像捕捉、室内结构和布局感知等工作，以协助投影设备进行颜色和畸变校正，让投影出的图像看起来更真实。

主图像还是会在房间中放置的电视等传统显示设备上显示，而该系统投影的环境图像分辨率会比主画面低，但并不会影响用户体验。此外，投影的图像未必就只是墙上显示的二维画面，可以通过 3D 眼镜的形式获得更逼真的效果。

欧洲研究人员开发的 CEEDs 项目利用一种沉浸式、多模式的互动系统，通过虚拟现实技术，使用户能够“身临其境地体验”大数据，这一系统甚至能够参照被试者的潜意识，对可视化场景进行调整。

该项目创立了一种沉浸式、多模式的互动系统，即经验归纳机器（XIM）。一旦用户开始佩戴 XIM 系统的虚拟现实耳机，集合数据就会以不同形状或形式展现，这有助于降低理解数据的难度。这种可视化场景如同镜子一般，能够随着用户的反应而发生改变。如果有

趣的事物吸引了用户的注意，场景的焦点也会随之调整。有趣的是，虚拟场景参考的不一定是用户有意识的反应，研究者正在寻找场景可随潜意识变化而变化的线索。

CEEDs 项目使用可穿戴技术来测量人们对可视化后的大数据的反应。XIM 系统通过各种设备监控手势、眼球运动或心率。运动传感器对姿势和身体动作进行追踪。手套记录手部动作，以及测量握力和皮肤反应。语音设备检测用户说话时的情感特征。此外，系统对面部表情、瞳孔扩张和其他参数进行测量，以调整大数据的呈现方式。

15.4 智能化

人工智能已经发展了几十年的时间，但在近些年的发展却比较缓慢。大数据的出现又为人工智能提供了较多的用武之地，唤醒了人工智能的巨大潜力。而人工智能已经存在的理论和方法，也逐渐被用于大数据技术中，并取得了一定的成果。大数据与人工智能正在相辅相成，共同呈现加速发展的趋势。

15.4.1 传统人工智能

目前，常用的人工智能方法包括机器学习、人工神经网络、智能搜索、知识表现、模糊逻辑等。想要使这些算法尽可能地发挥其优势并不容易。在人工智能的早期，并没有足够大的数据量和足够强大的计算能力，人工智能并没有得到较好的发展。

例如，人工神经网络在 20 世纪 40 年代就出现了。人工神经网络又称连接机模型，是在现代神经学、生物学、心理学等学科研究的基础上产生的。1986 年，Rumelhart, Hinton, Williams 发展了 BP (Back Propagation) 算法，掀起了基于统计模型的机器学习热潮。神经网络已经成为模式识别的强有力的工具。其分类功能特别适合于分类的应用。BP 算法是应用最多的一种神经网络形式。BP 神经网络具有较强的自学习和自适应能力、非线性映射能力、泛化能力以及容错能力，因此国内外不少研究学者都对其进行了研究，并解决了不少应用问题。然而，由于基于 BP 算法的人工神经网络的理论分析难度大，其训练效果依赖于使用者的经验与技巧，且容易过拟合，以及训练速度过慢等缺点，人工神经网络在经历了十多年的高潮期后，又遇到了瓶颈。

机器学习是近三十年来才兴起的一门多领域交叉学科，它涉及概率论、统计学、逼近论、凸分析、算法复杂度理论等多门学科。它专门研究计算机怎样模拟或实现人类的学习行为，以获取新的知识或技能，重新组织已有的知识结构，使之不断改善自身的性能。机器学习是人工智能的核心，也是计算机实现智能的根本途径。机器学习的应用范围很广，如在线广告、垃圾邮件过滤、手写识别、机器翻译等。但机器学习本身也有缺陷，容易引起维数灾难和过拟合。经典的机器学习不能真正表达“学习”的过程，无法产生具有确切现实意义的事物的概念，比如人脸识别，其实机器并没有得到“人脸”真正的实际意义，

只是把人脸与其他事物区分开来。

除了以上提到的几个人工智能方法以外,深度学习正在成为一个重要的人工智能分支。深度学习是机器学习研究中的一个新领域。其目的在于建立和模拟人脑进行分析学习的神经网络。一些学者认为深度学习是建立真正的人工智能的正确方向。深度学习有监督学习和无监督学习之分。较之其他的人工智能方法,深度学习方法需要的人工协助较少,在预设较少的情况下,也可以达到较好的效果,适合已知条件不多的情况。目前对于深度学习的研究还有较多的工作要做。在深度学习模型方面是否还有更好的学习模型是一个可研究的方向。另外,有效的并行化训练学习也是一个非常重要的研究方向。对于深度学习方法的研究,也曾遇到过瓶颈。深度学习算法对于计算的时间和资源要求比较高,在解决现实问题时表现出较多的弊端。而 GPU 技术的发展,再次把深度学习带进研究者的视野,其所需的计算时间和计算资源不再是一个重要问题。然而单个 GPU 对大规模数据识别或相似任务数据集并不适用,如何充分利用深度学习来增强传统学习算法的性能仍是目前各个领域研究的重点。

总体来看,在整个 IT 行业的存储能力、计算能力和通信能力都迅速发展的情况下,人工智能领域已经取得了较多的突破。人工智能的研究成果已经在各行各业得到了广泛的应用,如数据挖掘、语音识别、工业机器人、银行业软件、医疗软件等。但是,目前的人工智能方法都只能处理已预先定义好的问题,实现既定的目标。一旦遇到未定义的情况,人工智能便束手无策。因此,现阶段的人工智能技术并不能使机器具有真正的自主学习和研究的能力,更无法奢谈拥有创造能力。而使机器获得学习能力、研究能力和创造能力,恰恰是人工智能技术发展的目标。

15.4.2 基于大数据的人工智能

传统的人工智能虽然已经取得了较多的成果,但或多或少存在着计算时间和计算资源等方面的瓶颈。而大数据技术的发展,为人工智能的发展提供了良好的契机。大数据技术使解决人工智能的扩展性和成长性问题成为可能。由于数据量和计算资源的限制,以往的人工智能技术不能发展出与人类相似的学习能力、研究能力和创造能力。人工智能是一件很复杂的事情,产生人工智能需要海量数据和对这些海量数据的超级处理能力,而以前的机器所得到的数据量和拥有的数据处理能力都是不够的。

人工智能的发展,正如人本身一样,需要学习大量的知识和经验,这些知识和经验需要海量的数据作为支持。大数据技术的发展,为分析和储存海量的数据提供了技术支持,使得机器得到的数据量和机器拥有的数据处理能力,与形成人工智能所需要的数据量和数据处理能力不匹配的矛盾得到了缓解。在这种情况下,人工智能的理论、方法和技术的巨大潜力才有可能被真正地逐步释放出来,实现人工智能的发展目标,并反过来进一步推动大数据技术的发展,形成有效的相互推动。

目前基于大数据的智能分析已经有较多的应用。在企业计算业务领域,大数据可以提

供智能组织支持，提升决策、管理的效率。业界有的企业已经定义了下一代产品形态，即企业大数据分析引擎，关注流化数据处理和非结构化的数据处理。这个引擎能帮助企业垂直行业市场，进一步加强与用户的紧密联系，从而在部署服务战略上走得更远。

海量数据对金融、运营商等行业客户业务的智能分析提出了新的挑战。到目前为止，大数据技术已能够对一些数量巨大、种类繁多、价值密度极低、本身快速变化的数据进行有效和低成本的存取、检索、分类、统计。然而，如何能够同样有效和低成本地对收集和拥有的大数据进行智能分析，从而充分挖掘大数据的经济价值和社会价值，是大数据技术面临的一大难题。

在大数据智能推荐应用方面，也取得了较多的成果。例如，Netflix 的影片推荐系统、Facebook 的社交图谱、Amazon 的购物推荐系统等，已经依靠深度学习和其他人工智能方法，实现了大数据之上的巨大商业价值。Google 还对大数据的机器深度学习和建立知识树 Knowledge Graph 投入巨大的研究资源，希望能够对人们日常生活中所普遍关心的问题了解答。

2012 年 6 月，斯坦福大学吴恩达教授带领团队研发的 Google 大脑项目获得了巨大成功。该项目使用了 16000 个 CPU 核的并行计算平台，训练一种具有 10 亿个节点，被称为“深度神经网络”的机器学习模型。该项目直接把海量数据输入系统，系统会自动从数据中学习。该模型在 YouTube 的海量视频数据中自动搜索“猫”，获得了成功。该团队因此获得了 Google 公司 4700 万美元的资金资助。

在智能设备方面，苹果公司语音助手 Siri 的发布，引领了智能手机交互方式上的新变革。Siri 创立于 2007 年，Siri 技术来源于美国国防部高级研究规划局所公布的 CALO 计划，具有学习、组织以及认知能力。2010 年 Siri 被苹果以 2 亿美元收购，最初以文字聊天服务为主，随后通过与全球最大的语音识别厂商 Nuance 合作，实现了语音识别功能，随后衍生出了民用版软件 Siri 虚拟个人助理。虽然 Siri 存在着一定的不足，但它预示着智能设备发展的最重要的趋势。

微软推出的智能个人助手 Cortana 基于 Bing 存储的大数据为用户提供智能的服务。Cortana，中文名为微软小娜。小娜作为微软“革命性”的助手技术，与 Siri 相比提供了一种不同的呈现方式。微软对 Cortana 的描述为“你手机上的私人助手，为你提供设置日历项、建议、进程等更多帮助”，它能够和用户进行交互，并且尽可能模拟人的说话语气和思考方式跟用户进行交流（图 15.4）。



图 15.4 微软智能个人助手 Cortana

IBM公司的超级电脑沃森(Watson)系统是基于大数据技术的智能交互的另一个成功案例。它使用了自然语言语义分析、信息提取、知识表现、自动化推理、机器学习等人工智能方法,是当代人工智能研究的代表性成就。2011年2月16日,沃森在美国最受欢迎的智力竞猜电视节目《危险边缘》中以绝对优势击败了两名人类冠军。沃森最终答题总成绩高达77147分,远高于两位人类选手21600分和24000分的成绩,勇夺100万美元大奖。沃森拥有比它的上一代产品“深蓝”更强大的运算能力,能应对更复杂的比赛规则。现今,沃森已被用于医学中的癌症诊断,同样也具备比最有经验的医生更高的准确率。但目前机器只是在某些特定领域表现出和人类可比的智力水平,还有很多领域机器还差得很远。虽然沃森使用了机器学习的技术,已经具有一定的学习能力,不过这个学习还是有指导的,完全的自学习能力还有待进一步研究和开发。

不难看出,已有的人工智能技术已经能够使大数据的使用价值凸显出来,初步展现大数据的价值。但其实,人工智能的潜力还远远未被释放出来。建立具有真正意义的人工智能系统,是人类一直以来的梦想。面向大数据和人工智能的研究近来呈现出螺旋上升式发展态势,大数据时代的到来,赋予人工智能新的起点、新的使命和新的召唤。

在未来,计算机将有可能模仿人类的视、听、触、味、嗅5种感觉,在大数据的感知环境下,更好地帮助人类认知,对人们的生活和行业发展产生深远影响。很快,计算机将能够“看到”。科学家们相信,在未来5年内,系统不仅能够看到和识别可视数据的内容,而且能把像素转化为含义,开始像人类观看和解析图片那样从中理解其意义。在计算机视觉提升方面,未来,“类似人脑”的能力将使计算机能够分析颜色、纹理、材质或边缘信息,并能通过解析图像得出图片的意义。这将对医疗、零售、农业等行业产生深远影响。

计算机将能够“听到”。比如,它们能探测到可能有死亡征兆的树木的活动,提示相关人员在树木倒掉之前修剪或者砍伐这些树木,从而保护人们的安全与财产。再如,它们能探测到火灾中风向的改变,帮助消防人员确定后续行动,从而控制火灾。

此外,计算机还将能听到并理解对我们至关重要的声音。例如,IBM与医疗专家和学术机构合作收集数据,将婴儿声音与身体内部状况和行为关联起来,并且开发了先进的翻译系统,将来会在婴儿和幼儿身上使用。这种工具将识别并理解婴儿的咿呀学语,并根据学到的声音含义分析他们真正想表达什么,这样人们就可以知道婴儿是饥饿、过热、疲劳还是难受。

计算机将拥有触觉。从本质上来讲,触觉是一种物理体验。但是,借助红外线和触觉反馈技术,人们已经开始在游戏行业中模拟触觉。通过振动形式重新创造一种触感,并且用在移动设备和游戏机上,玩家能够在赛车游戏中获得驾驶感受。同样,一旦计算机拥有触觉,在线购物时,商家将使用触觉技术让客户在购买之前“触摸”商品。购物者在有衣物图像的屏幕上滑动手指,即可感受到衣物的质地。

拥有强大嗅觉功能的计算机可以让人们感觉更安全,例如计算机可以探测到全球主要城市的空气污染等级。此外,计算机也可以安放在艺术馆中,嗅出人的鼻子无法感觉到但有可能破坏重要艺术品的气体。在未来5年内,计算机或手机中嵌入的微型传感器将“嗅

出”用户是否患有感冒或其他疾病。

同时，认知系统时代的计算机将具备一套更加智能的系统，该系统一直处于不断学习和提高的状态。在认知系统时代，拥有人类五感的计算机将不再局限于演绎推理，或者从更普遍的数据中得出结论，而是模拟人类的归纳推理能力，同时也具备学习的能力。

当然，科学家们并不期望计算机完全替代人的功能。IBM 认为，认知型计算机的真正成功，并不在于它替代人脑的功能，而在于它提供的创新能够为人们带来更好的生活质量，而且为人们应对最严峻的挑战提供关键的信息，使人们能够提出创新的解决之道。认知型计算机的诞生，即是为了令人类和计算机在认知系统时代强强联合，完成更优秀的工作。

大数据与人工智能的交叉是未来计算机领域探索的一个重要方向，技术的交叉与整合是互联网未来发展的重要趋势之一。大数据与人工智能的研究相互交叉促进，产生了很多新的方法、应用和价值。大数据的发展本身使用了许多人工智能的理论和方法，人工智能也因大数据技术的发展进入了一个新的发展阶段。相信在大数据和人工智能的相互促进下，人工智能技术将有可能使机器真正获得自主学习和研究的能力，能够处理未预先定义的新的情况，并使其变成机器拥有的一种新知识，甚至演进出机器的创造能力，真正实现与人类相似甚至超越人类的智能。因此，大数据时代的到来，也正在开启人工智能的新篇章。

15.5 练习题

1. 大数据技术发展都有哪些趋势？
2. 举一个大数据在人工智能领域的具体应用案例。

参考文献

- [1] <http://www.yicai.com/news/2012/09/2072281.html>
- [2] <http://www.zcom.com/article/125312/>
- [3] <http://www.cnblogs.com/xuybin/p/3966022.html>
- [4] <http://tech.qq.com/a/20141118/002193.htm>
- [5] <http://www.csdn.net/article/2014-10-10/2822012>
- [6] <http://www.36kr.com/p/152405.html>

第16章

知名企业大数据架构简介

16.1 腾讯

16.1.1 背景介绍

腾讯是目前中国最大的互联网综合服务提供商之一，也是中国服务用户最多的互联网企业之一。首先，即时通信软件 QQ，最高同时在线账户数达到 2.06 亿，月活跃账户数更是高达 8.29 亿，QQ 智能终端月活跃账户数达到 5.21 亿。其次，“微信和 WeChat”合并月活跃账户数达到 4.38 亿，QQ 空间月活跃账户数达到 6.45 亿，QQ 空间智能终端月活跃账户数达到 4.97 亿。从这些数据可以看到，腾讯每天的数据量是异常庞大的，目前最高日接入消息数为 10000 亿条，日接入数据量为 200TB，并发分拣业务接口为 10000 个。

腾讯在设计架构时主要考虑了 3 个主要的需求。

① 数据开放：使得公司数据集中形成数据开放，在保障数据安全性的前提下，提供自助化服务平台，满足快速增长的需求。

② 专业化：从提供大量独立的系统/工具转变为提供集成、一体化、自动化数据开发平台服务。对来源于各个业务块的数据进行整合和深入挖掘产生用户画像，为业务提供有价值的服务，并且快速孵化更多的数据应用。

③ 成本与性能：优化平台存储和计算方案，优化数据模型和算法，去除重复计算和存储；通过建设大规模集群，形成规模效应，提升平台能力并降低成本；随着平台上的数据

量、用户数、任务数不断增长，每个新用户/新任务带来的新增成本不断降低，成本优势可以不断放大。

16.1.2 整体架构

从图 16.1 可以看出，腾讯大数据平台主要由 4 个核心模块构成：TDW、TRC、TDBank 和 Gaia。从下至上，Gaia 负责腾讯大数据平台上所有集群资源的调度管理工作，为上层的业务和数据服务提供必要的计算资源。构建于 Gaia 之上的两个部分 TDW 和 TRC 分别负责批量的离线计算和流式的实时计算。TDBank 则作为统一的数据采集入口。

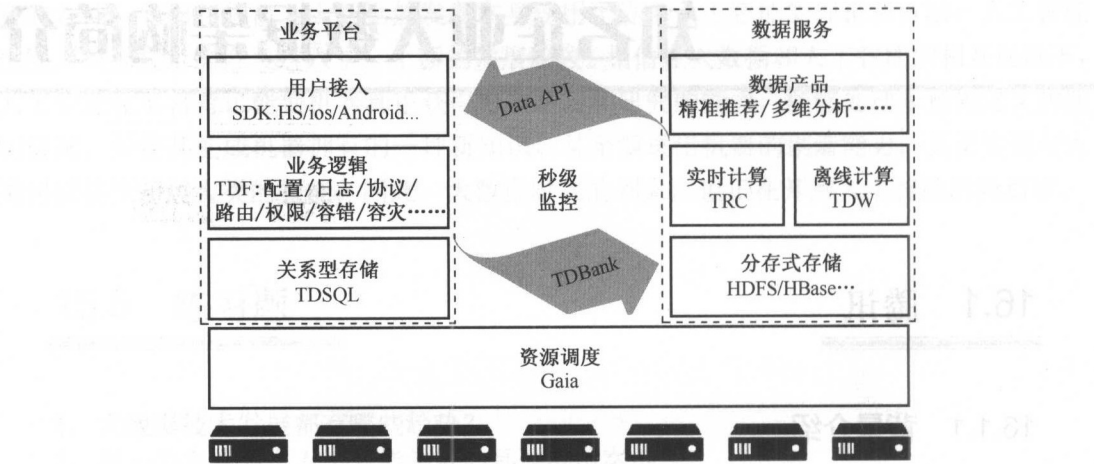


图 16.1 腾讯大数据平台的整体架构

1. Gaia

作为腾讯大数据平台的统一资源调度平台，Gaia 极大地简化了开发者管理计算资源的工作，使得整个集群对于开发者而言更像一个逻辑上的超级计算机。Gaia 提供高并发任务调度和资源管理，实现集群资源共享，具有很高的可伸缩性和可靠性，这使得它支持各种不同的计算任务和业务，甚至包括在线 Service 业务（图 16.2）。

为了支撑大规模的集群，在开源的 Hadoop YARN 之上，Gaia 自主研发了更强大的调度器 Sfair。Sfair 优化了调度逻辑，提供更好的可扩展性，并进一步增强调度的公平性，提升可定制化，将调度吞吐量提升了 10 倍以上。为了使上层多样化的计算框架稳定运行，Gaia 除了对 CPU 和内存的资源管理之外，还增加了 Network IO，Disk Space，Disk IO 等资源管理维度，提高了隔离性，为业务提供了更好的资源保证和隔离。

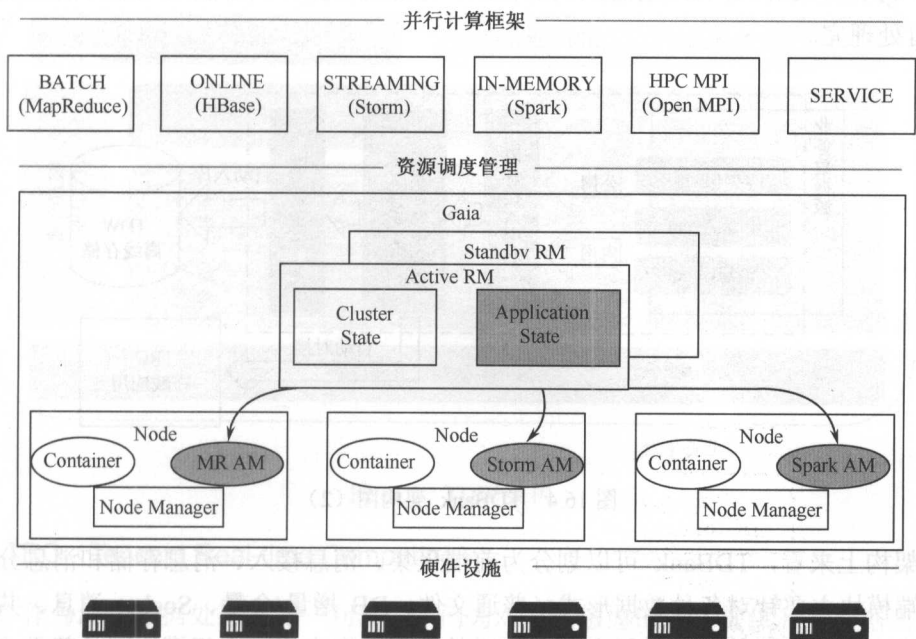


图 16.2 Gaia 架构

2. TDBank (Tencent Data Bank)

TDBank 是从业务数据源端实时采集数据，进行预处理和分布式消息缓存后，按照消息订阅的方式，分发给后端的离线和在线处理系统（图 16.3 和图 16.4）。

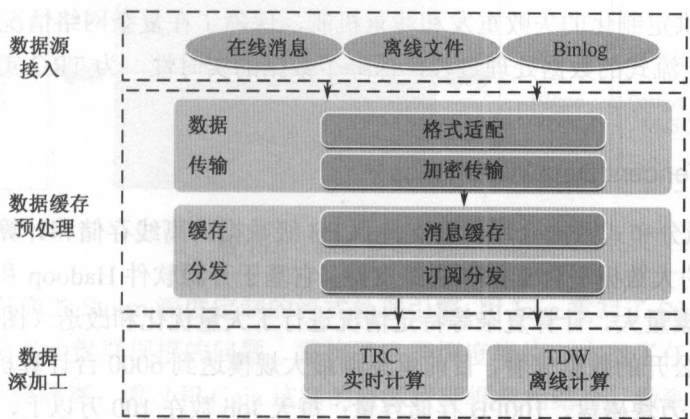


图 16.3 TDBank 架构图（1）

它构建了数据源和数据处理系统间的桥梁，它将数据处理系统与数据源解耦，为离线计算 TDW 和在线计算 TRC 平台提供数据支持。目前通过不断改进，将以前 Linux+HDFS

的模式，转变为集群+分布式消息队列的模式，将以前需要一天才能处理完的消息量缩短到 2 秒即可处理完。

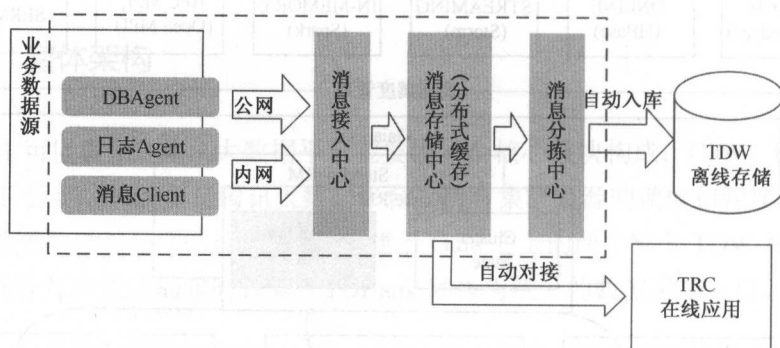


图 16.4 TDBank 架构图 (2)

从架构上来看，TDBank 可以划分为前端采集、消息接入、消息存储和消息分拣等模块。前端模块主要针对各种数据形式（普通文件、DB 增量/全量、Socket 消息、共享内存等）提供实时采集组件，提供了主动且实时的数据获取方式。中间模块则是基于“发布—订阅”模型的分布式消息中间件，它起到了很好的缓存和缓冲作用，避免了因后端系统繁忙或故障而导致的处理阻塞或消息丢失。针对不同应用场景，TDBank 提供数据的主动订阅模式，以及不同的数据分发支持（分发到 TDW 数据仓库、文件、DB、HBase、Socket 等）。整个数据通路透明化，只需要简单配置，即可实现一点接入，整个大数据平台可用。

另外，为了减少大量数据进行跨城网络传输，TDBank 在数据传输的过程中进行数据压缩，并提供公网/内网自动识别模式，极大地降低了专线带宽成本。为了保障数据的完整性，TDBank 提供定制化的失败重发和滤重机制，保障了在复杂网络情况下数据的高可用性。TDBank 基于流式的数据处理过程，保障了数据的实时性，为 TRC 实时计算平台提供实时的数据支持。

3. TDW (Tencent Data Warehouse)

TDW 是腾讯分布式数据仓库。它支持百 PB 级数据的离线存储和计算，为业务提供海量、高效、稳定的大数据平台支撑和决策支持。它基于开源软件 Hadoop 和 Hive 进行构建，并且根据公司数据量大、计算复杂等特定情况进行了大量优化和改造（图 16.5）。

从腾讯目前公开的数据来看，目前单集群最大规模达到 6000 台计算机、14 万核 CPU、380TB 内存、7.2 万块磁盘、100PB 存储容量；每天 Job 数在 100 万以上，每天扫描数据量为 6.5 PB，存储利用率为 85%，CPU 利用率在 90%以上，网络利用率在 90%以上。

同时为了满足挖掘分析与交互式实时查询的计算需求，腾讯大数据使用了 Spark 平台来支持挖掘分析类计算、交互式实时查询计算以及允许误差范围的快速查询计算。目前腾讯大数据拥有超过 200 台计算机的 Spark 集群，并独立维护 Spark 和 Shark 分支。

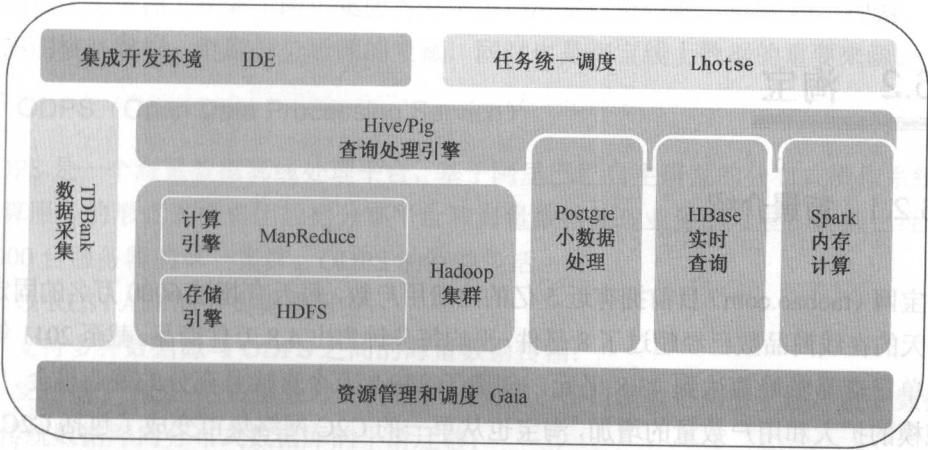


图 16.5 TDW 架构

4. TRC (Tencent Real-time Computing)

TRC 作为海量数据处理的另一利器，专门为对时间敏感的业务提供海量数据实时处理服务。通过海量数据的实时采集、实时计算，实时感知外界变化，从事件发生到感知变化再到输出计算结果，整个过程在秒级时间内完成（图 16.6）。

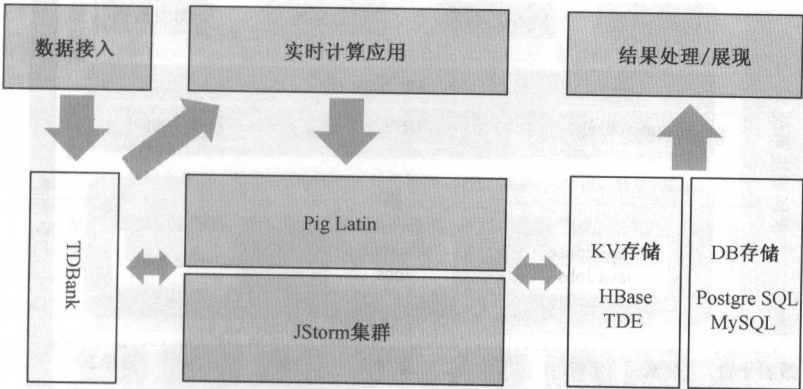


图 16.6 TRC 整体架构

TRC 是基于开源的 Storm 深度定制的流式处理引擎，用 Java 重写了 Storm 的核心代码。为了解决资源利用率和集群规模的问题，重构了底层调度模块，实现了任务级别的权限管理、资源分配、资源隔离，通过和 Gaia 这样的资源管理框架相结合，做到了根据线上业务实际利用资源的状况，动态扩容和缩容，单集群轻松超过 1000 台规模。为了提高平台的易用性和可运维性，提供了类 SQL 和 Pig Latin 这样的过程化语言扩展，方便用户提交业务，提升接入效率，同时提供系统级的指标度量，支持用户代码对其扩展，实时监控整个系统运营环节。另外将 TRC 的功能服务化，通过 REST API 提供 PaaS 级别的开放，用户无须了解底层实现细节就能方便地申请权限、资源和提交任务。

16.2 淘宝

16.2.1 背景介绍

淘宝网(taobao.com)目前拥有近5亿的注册用户数,每天有超过6000万名的固定访客,同时每天的在线商品数已经超过了8亿件,平均每分钟售出4.8万件商品。截至2011年年底,淘宝网单日交易额峰值达到43.8亿元,创造了270.8万个直接且充分的就业机会。随着淘宝网规模的扩大和用户数量的增加,淘宝也从单一的C2C网络集市变成了包括C2C、团购、分销、拍卖等多种电子商务模式在内的综合性零售商圈。目前它已经成为世界范围的电子商务交易平台之一,因此在其架构中面临着大量高性能问题。

16.2.2 整体架构

淘宝数据平台架构图如图16.7所示。

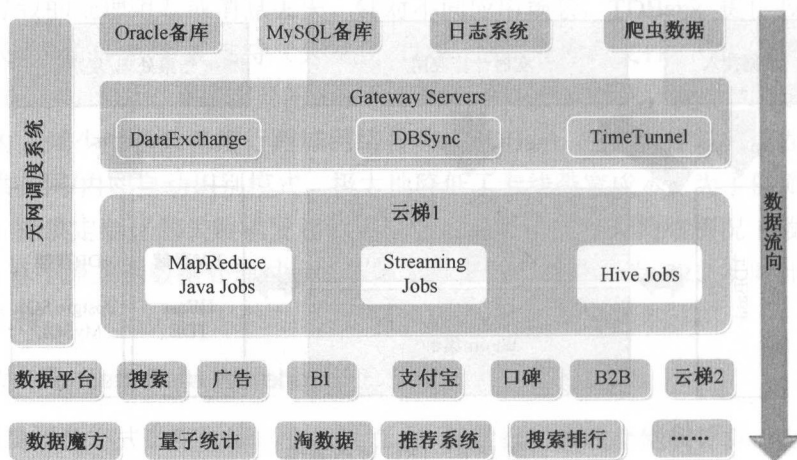


图 16.7 基于云梯的淘宝数据平台架构图

1. 云梯

云梯是阿里集团最大的Hadoop集群,机器规模接近9000台,是国内第一个超大规模的跨机房集群,存储容量超过200PB,单集群每日运行的作业数超过25万。云梯除了支持MapReduce、Hive、Mahout等计算框架外,还支持MPI、Spark、R等新兴的离线计算模型。云梯在Hadoop社区版本基础上做了大量功能性、稳定性、可扩展性方面的改进,有很多已

经贡献给社区。云梯上积累了阿里集团各家子公司海量业务数据，包括 BI、搜索、广告等部门产出的核心数据，是数据分析师的宝藏，同时也是淘宝线上数据的重要来源。

2. ODPS (Open Data Processing Service)

ODPS 是一个海量数据离线处理平台，基于阿里巴巴自主研发的分布式操作系统开发，以云计算服务的形式支撑集团数据分享平台和海量数据处理业务的发展。目前已经部署了超过 5000 台服务器的单一集群。ODPS 的特性包括：

- ① 以 REST API 的方式提供服务；
- ② 支持多种数据源与 ODPS 之间的海量数据传输；
- ③ 支持基于 SQL 的数据处理，并提供块编程、窗口函数等高级功能，便于数据分析人员从传统数据库向分布式数据库的平滑迁移；
- ④ 支持经过扩展的 MapReduce 编程框架，让大数据编程变得更加简单和高效；
- ⑤ 支持 SVD 等常用矩阵算法，以及多项海量数据挖掘功能；
- ⑥ 提供了用户空间、多用户协同、数据授权与共享、作业管理、基于依赖关系的作业调度、事件注册与通知、监控与报警等多种功能，用户基于 ODPS 可以打造完整、强大的数据仓库解决方案。

淘宝大数据计算框架如图 16.8 所示。

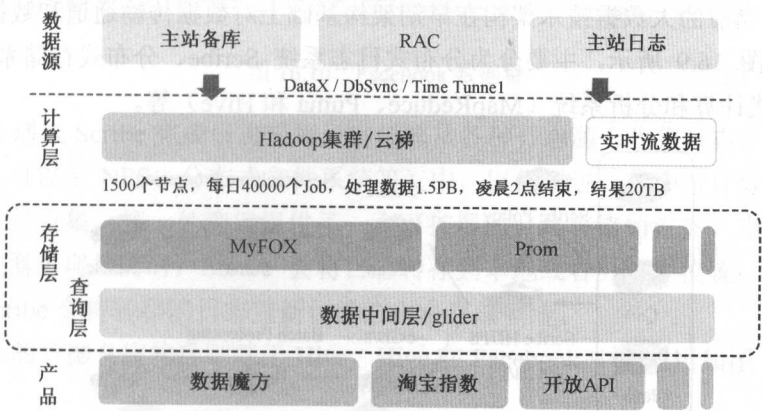


图 16.8 淘宝大数据计算框架

16.3 Facebook

16.3.1 背景介绍

Facebook 一直是大数据技术最积极的应用者，因为它拥有的数据量极其巨大，一份资

料显示 2011 年它拥有的压缩数据已经有 25PB，未压缩数据为 150PB，每天产生的未压缩的新数据有 400TB。在 Facebook，大数据技术被广泛应用在广告、新闻源、消息/聊天、搜索、站点安全、特定分析、报告等各个领域。Facebook 也是 Apache 大数据开源项目的最大贡献者之一。Facebook 于 2007 年前后正式转向 Hadoop 计算框架，随之它向 Apache 基金会贡献了大名鼎鼎的 Hive、ZooKeeper、Scribe、Cassandra 等开源工具，当前 Facebook 的开源进程仍在积极推进着。

16.3.2 整体架构

Facebook 早期的大数据技术架构是建立在 Hadoop、HBase、Hive、Scribe 等开源工具基础上的。日志数据流从 HTTP 服务器产生，通过日志收集系统 Scribe 耗费秒级时间传送到共享存储 NFS 文件系统，然后通过小时级的 Copier/Loader（即 MapReduce 作业）将数据文件上传到 Hadoop。数据摘要通过每天例行的流水作业产生，它基于 Hive 的类 SQL 语言开发，结果会定期更新到前端的 MySQL 服务器，以便通过 OLTP 工具产生报表。Hadoop 和 Hive 集群由 3000 台服务器组成，每台服务器有 8 核、32GB 内存、12TB 硬盘，能够很好地解决扩展性和容错性方面的问题，但是早期系统的主要问题是整体的处理延迟较大，从日志产生起 1~2 天后才能得到最终的报表。

Facebook 当前的大数据技术架构在早期架构基础上对数据传输通道和数据处理系统进行了优化，如图 16.9 所示，主要分为分布式日志系统 Scribe、分布式存储系统 HDFS 和 HBase、分布式计算和分析系统（MapReduce、Puma 和 Hive）等。

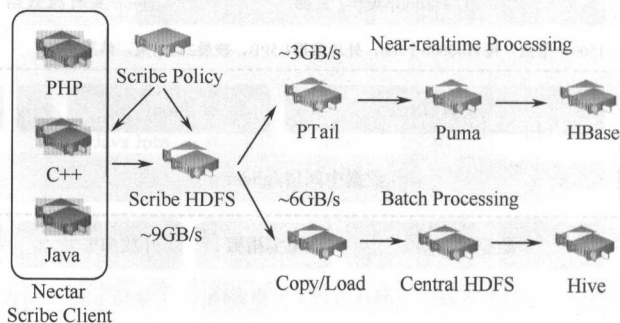


图 16.9 Facebook 大数据技术架构

其中，Scribe 日志系统用于聚合来自大量 HTTP 服务器的日志数据。Thrift 是 Facebook 提供的软件框架，用于跨语言的服务开发，能够在 C、Java、PHP、Python 和 Ruby 等语言之间实现无缝支持。采用 Thrift RPC 来调用 Scribe 日志收集服务进行日志数据汇总。Scribe Policy 是日志流量和模型管理节点，将元数据传送给 Scribe 客户端和 Scribe HDFS，采集的日志数据存储在 Scribe HDFS。Facebook 对早期系统优化后的数据通道称为 Data Freeway，能够处理峰值为 9GB/s 的数据，并且端到端的延迟在 10s 以内，支持超过 2500 种日志。Data

Freeway 主要包括 4 个组件：Scribe、Calligraphus、Continuous Copier 和 PTail。Scribe 用于客户端，负责通过 Thrift RPC 发送数据；Calligraphus 在中间层梳理数据并写到 HDFS，它提供了日志种类的管理，利用 ZooKeeper 进行辅助；Continuous Copier 将文件从一个 HDFS 复制到另一个 HDFS；PTail 并行地 tail 多个 HDFS 上的目录，并写文件数据到标准输出。在当前架构中，一部分数据仍然以批处理的方式通过 MapReduce 进行小时级的处理，存储在中央的 HDFS，每天通过 Hive 进行分析处理。另一部分接近实时的数据流则通过 Puma 来进行分钟级的处理。Facebook 对专门分析提供 Peregrine (Hpal) 工具，对周期性分析提供 Nocron 工具。

Facebook 使用 HBase 存放消息，消息数据模型特点如下：

- ① 对于中小型数据，使用 HBase，如消息元数据、搜索索引和小消息内容等；
- ② 对于附件或大的消息数据，使用 HayStack，如图片和视频等。

Facebook 数据量如图 16.10 所示。

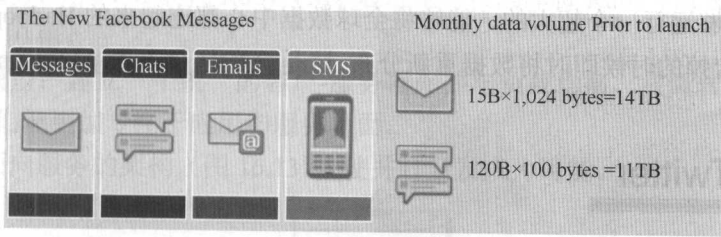


图 16.10 Facebook 数据量

Facebook 通过 Scribe 完成日志采集。它能够从各种日志源上收集日志，存储到一个中央存储系统（可以是 NFS、分布式文件系统等）中，以便于进行集中统计分析处理。它为日志的“分布式收集，统一处理”提供了一个可扩展的、高容错的方案。当中央存储系统的网络或者机器出现故障时，Scribe 会将日志转存到本地或者另一个位置；当中央存储系统恢复后，Scribe 会将转存的日志重新传输给中央存储系统。

Facebook 的数据仓库构建则基于 Hive，Hive 在 Facebook 中的应用如图 16.11 所示。

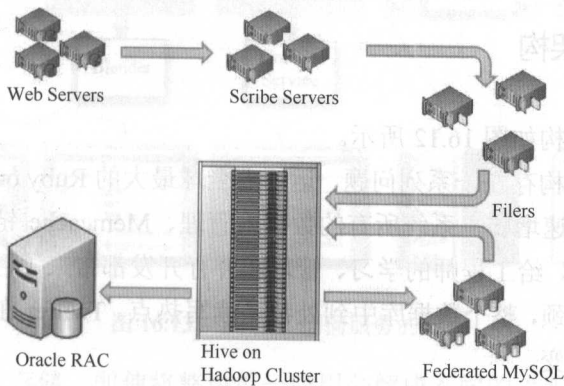


图 16.11 Hive 在 Facebook 中的应用

Hive 被 Facebook 用来实现用户数据统计，如点击数、浏览数、广告展示等。

16.3.3 技术架构展望

Facebook 未来的大数据技术架构的雏形已经显现。首先开源的是可能替代 Hadoop 系统中 MapReduce 的 Corona，类似于 Yahoo 提出的 YARN。Corona 最大的一个进步是其集群管理器做到了基于 CPU、内存和其他作业处理的需求资源的管理，这使得 Hadoop 集群的应用领域更加广泛。其次是 Facebook 最新的交互式大数据查询系统 Presto，类似于 Cloudera 的 Impala 和 Hortonworks 的 Stinger，满足了 Facebook 迅速膨胀的海量数据仓库快速查询需求。据 Facebook 称，使用 Presto 进行简单的查询只需要几百毫秒，即使是非常复杂的查询，也只需数分钟便可完成，它在内存中运行，并且不会向磁盘写入。第三是 Wormhole 流计算系统，类似于 Twitter 的 Storm 和 Yahoo 的 Storm YARN。第四个重要项目是 Prism，它能够运行一个超大的、能够将全球数据中心都连起来的 Hadoop 集群，可以在一个数据中心宕掉的时候即时将数据重新分布，这是一个与 Google 的 Spanner 类似的项目。

16.4 Twitter

16.4.1 背景介绍

Twitter 是一个广受欢迎的社交网络及微博客服的网站，允许用户将自己的最新动态和想法以移动电话中的短信息形式（推文）发布。Twitter 现在有 1.5 亿名全球活跃用户，查询率（Query Per Second，QPS）为每秒 30 万次，流量为 22 MB/s，系统每天处理 4 亿条推文数据，是互联网上访问量最大的十个网站之一。如此大的流量和数据必定需要一个高并发、高可用的技术架构来支撑。

16.4.2 整体架构

Twitter 的整体架构如图 16.12 所示。

Twitter 之前的架构存在一系列问题，运行着全球最大的 Ruby on Rails 应用。随着用户规模和服务数量的快速增长，系统所有的数据库管理、Memcache 链接以及公共 API 的代码属于同一个代码库，给工程师的学习、管理和并行开发都带来了巨大困难。MySQL 存储系统已经遇到性能瓶颈，整个数据库中到处都是读写热点。Twitter 在此基础上重建了架构，做出了以下一系列改变。

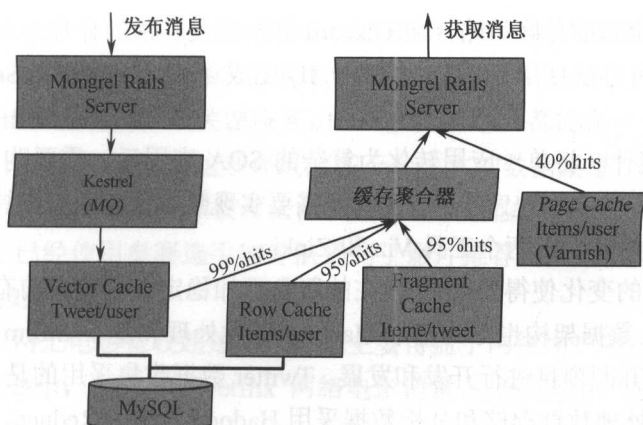


图 16.12 Twitter 的整体架构

① 前端服务：用 JVM 取代 Ruby VM。通过重写代码库将 Ruby VM 服务移植到 JVM，使性能提高了 10 倍。

② 编程模型：建立一个统一的客户-服务器库并与负载均衡、故障转移策略等绑定，从而让工程师们能更加专注于应用和服务界面。

③ 采用面向服务的架构（图 16.13），使并行开发成为可能。

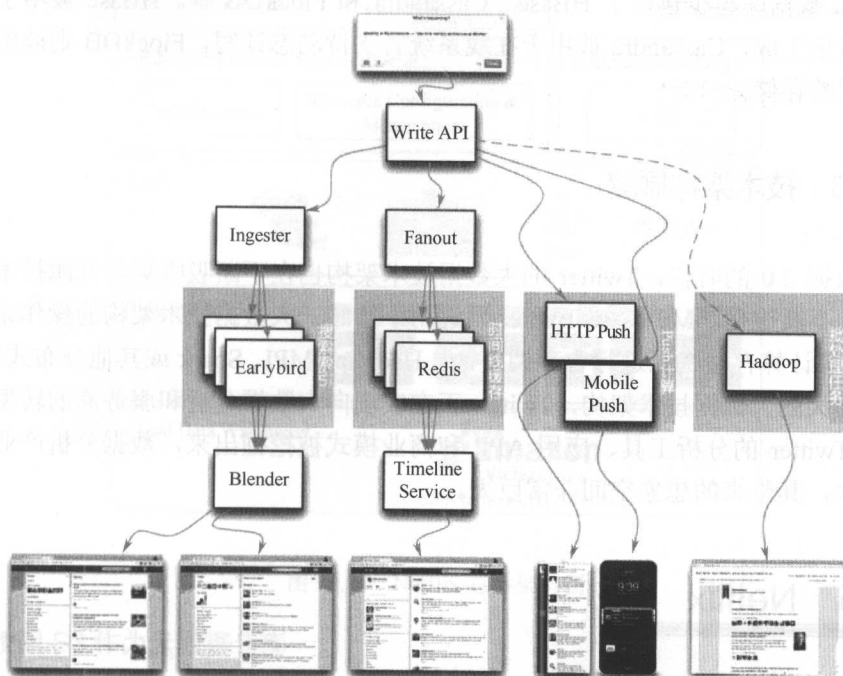


图 16.13 Twitter 面向服务的架构

④ 推文的分布式存储。即使将整块单一应用分解成不同的“服务”，存储依然是个巨大的瓶颈。过去 Twitter 采用的单一 MySQL 主数据库只能线性写入推文，Twitter 决定在推

文的存储上采用全新的分区策略，用 Gizzard 框架创建容错的分片分布式数据库存储推文，但这样一来就没有办法使用 MySQL 的唯一 ID 生成功能。Twitter 用 Snowflake 解决了这个问题。

⑤ 监测与统计。将单一应用转化为复杂的 SOA 应用后，需要购买匹配的工具才能够驾驭。Twitter 的服务推出速度很快，同时还需要实现数据化的决策支持，Twitter 的 Runtime 系统团队为工程师开发了两个工具 Viz 和 Zipkin。

这些架构上的变化使得 Twitter 现在的服务更加稳定，相比之前有了很大的改进。

Twitter 的大数据架构也分为基于 Hadoop 的批处理和基于 Storm 的实时流计算等主要类型，主要基于开源项目进行开发和发展。Twitter 数据收集采用的是 Facebook 开源的日志工具 Scribe，批处理数据存储和分析数据采用 Hadoop 和 MapReduce，在大数据上的快速分析采用 Pig。Pig 是基于 Hadoop 的并行计算高级编程语言，它提供一种类 SQL 的数据分析高级文本语言，称为 Pig Latin，该语言的编译器会把类 SQL 的数据分析请求转换为一系列经过优化处理的 MapReduce 运算。Pig 支持的常用数据分析主要有分组、过滤、合并等。

Storm 是 Twitter 的开源流计算平台，Storm 通过简单的 API 使开发者可以可靠地处理无界持续的流数据，进行实时计算，开发语言为 Clojure 和 Java。Storm 的应用场景很多，如实时分析、在线机器学习、持续计算等。

Twitter 的存储工具有很多，体现了其在不同发展阶段的作用，也适用于不同的应用场景。NoSQL 数据库至少包含了 HBase、Cassandra 和 FlockDB 等。HBase 被用于批处理的分析和数据集生成，Cassandra 被用于在线系统且支持动态读写，FlockDB 则被用于实时分布的社交图的存储。

16.4.3 技术架构展望

在大数据 2.0 的时代，Twitter 的大数据技术架构也在不断吸收更多开源技术的精华，兼容并蓄，不断进化。Mesos 被 Twitter 引进用于分布式大数据技术架构的操作系统，能够对 Hadoop 等计算存储资源进行合理调度，如 Hadoop、MPI、Spark 或其他分布式计算框架。

基于强大的大数据技术架构，Twitter 正在实现向大数据分析和服务商的转型，越来越多的基于 Twitter 的分析工具、应用 APP 和商业模式被挖掘出来，数据分析产业链逐步建立起来，其带来的想象空间非常巨大。

16.5 Netflix

16.5.1 背景介绍

Netflix 是一家美国公司，在美国、加拿大提供互联网随选流媒体播放，以及定制 DVD、蓝光光碟在线出租业务。该公司成立于 1997 年，总部位于加利福尼亚州洛斯盖图，1999 年

开始提供订阅服务。2007 年 2 月 25 日，Netflix 宣布已经售出第 10 亿份 DVD。2009 年，该公司可提供多达 10 万部 DVD 电影，并有 1000 万名订户。

推荐引擎是 Netflix 公司的一项关键服务，1000 多万名顾客都能在一个个性化网页上对影片做出 1~5 分的评级。Netflix 将这些评级放在一个巨大的数据集里，该数据集容量超过了 30 亿条。Netflix 使用推荐算法和软件来标识具有相似品位的观众对影片可能做出的评级。几年来，Netflix 已经使用参赛选手的方法提高了影片推荐的效率，这已经得到了很多影片评论家和用户的好评。

2011 年 Netflix 网上电影营收超过苹果，这主要得益于网络用户对在线视频的强大需求。HIS 的一份报告显示，2011 年 Netflix 网络电影销量占据美国用户在线电影总销量的 45%。

16.5.2 整体架构

Netflix 选择把所有的数据存储于亚马逊的存储服务（S3）中，这也是架构得以实现的核心原则。总体架构如图 16.14 所示。

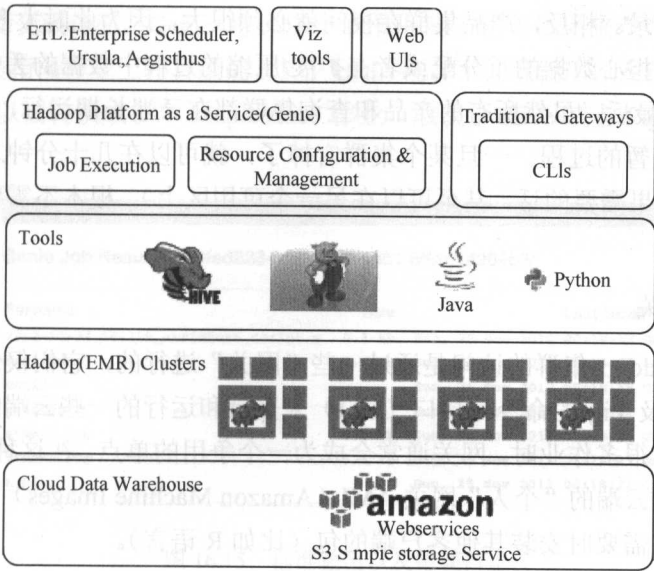


图 16.14 Netflix 总体架构图

1. 使用 S3 作为云数据仓库

S3 是 Netflix 基于云的数据仓库服务真正的“源”。所有值得保留的数据集都被存储在 S3 之中，包括很多数据流的信息，它们来自（拥有 Netflix 功能）电视机、PC 以及各种移动设备的使用过程，这些信息被称为 Ursula 的日志数据管道所抓取；同时还有来自 Cassandra 的维度数据。

那么为什么 Netflix 使用 S3 而不是 HDFS 作为“源”呢？首先，S3 提供了高达 99.999999999% 的持久性和 99.99% 的可用性（在特定的一年），能够承担两个设施中并发的数据丢失现象；其次，S3 提供了版本信息存储块，可以用它来防止意外的数据丢失，例如，一个开发人员错误地删除了一些数据，可以很容易地进行恢复；再次，S3 具有弹性，提供了几乎“无限”的规模扩展，这样数据仓库就实现了从几百 TB 到 PB 级的有序增长，而无需提前准备存储资源；最后，使用 S3 作为数据仓库可以帮助 Netflix 运行多个高动态的集群，这些适用于故障和负载。

另一方面，虽然 S3 的读/写速度比 HDFS 要慢，然而，大多数的查询和处理往往是多级的 MapReduce 作业。在第一阶段 Mapper 从 S3 平行地读取输入数据，Reducer 在最后阶段把输出数据返回至 S3，而 HDFS 和本地存储用于存储所有的中间级和临时数据，这就降低了性能的开销。

2. 针对不同工作负载的多个 Hadoop 集群

Netflix 目前使用亚马逊的 Elastic MapReduce，而把 S3 作为数据仓库可以针对不同的工作负载弹性配置多个 Hadoop 集群，所有的集群都连接相同的数据。

Netflix 每天都会动态地调整查询和产品集群，其实查询集群在夜间可以更小，因为那时很少有开发者登录。相反，产品集群在夜间就必须很大，因为此时大多数的 ETL 都在运行。Netflix 不需要担心数据的重分配或者在扩展/压缩的过程中数据的丢失现象，因为数据都分布在 S3 上。最后，虽然所有的产品和查询集群都在云端长期运行，但是 Netflix 可以把它们当做一个短暂的过程。一旦某个集群宕掉了，就可以在几十分钟之内启用另一个同等规模的集群（如果需要的话，甚至可以在另一个可用区上），根本不需要担心数据的丢失问题。

3. 工具及网关

Netflix 的 Hadoop 集群的访问是通过一些“网关”进行的，它们仅仅是开发者们通过 Hadoop、Hive 以及 Pig 的命令行接口（CLI）来登录和运行的一些云端的实例。当有很多开发者登录和运行很多作业时，网关通常会成为一个争用的单点。在这种情况下，鼓励“重量级”的用户启用云端的“个人”网关 AMI（Amazon Machine Images）的实例。使用个人网关允许开发者在需要时安装其他客户端的包（比如 R 语言）。

4. Genie——Hadoop 平台即服务

Amazon 提供了 Hadoop IaaS 平台，通过 Elastic MapReduce（EMR）提供服务。EMR 提供了 API 和 Hadoop 集群，在这里可以获取一个或多个 Hadoop 作业。Netflix 已经实现了 Hadoop PaaS 服务（也就是 Genie）；提供了一个更高级别的抽象，不需要构建新的 Hadoop 集群或者安装 Hadoop、Hive 以及 Pig 客户端，就可以通过 REST-ful API 提交单独的 Hadoop、Hive 或者 Pig 作业。此外它还允许管理员去管理和抽象云中不同后端的 Hadoop 资源配置。

Genie 是专为 Hadoop 生态系统定制的一组 REST-ful 服务集合，用于管理作业和资源。

有两个关键服务：Execution Service 和 Configuration Service。前者提供了 REST-ful API，用于提交和管理 Hadoop、Hive 以及 Pig 作业；后者是 Hadoop 资源的有效储存库，用于元数据的连接以及运行资源上的作业。

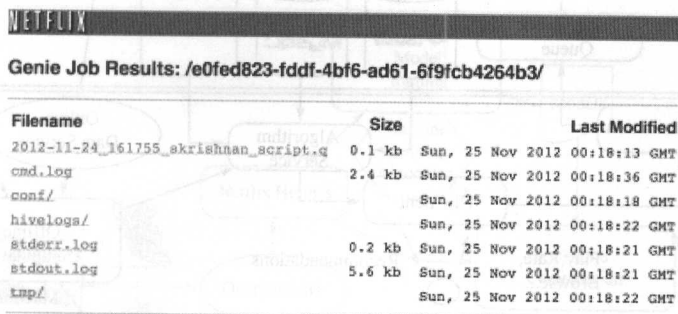
为什么要构建 Genie，而不是用一些已存在的东西呢？最简单的答案就是现在的开源社区中没有一个适合 Netflix 的需求——用于运行作业、后端集群的抽象、可以向不同集群提交作业、满足需求扩展度（横向或者纵向）的 API。

5. Execution Service

Execution API，负责客户端与 Genie 的交互。客户端通过向 Execution API 发送 JSON 和 XML 信息提交作业，其中包括的参数有：

- ① 作业的类型，Hadoop、Hive 或者 Pig；
- ② 作业的命令行参数；
- ③ 文件的依赖性，比如 S3 上的 scripts 和 jar 文件；
- ④ 时间表类型（比如“ad hoc”或者“SLA”），这样 Genie 就可以使用它来为作业映射适当的集群；
- ⑤ Hive 元存储需要连接的名称（比如 prod、test 或者一个设备名称）。

当一个作业提交成功后，Genie 将返回一个作业 id，它可以用来获得作业状态以及输出 URL。输出 URL 是指向作业工作目录的 HTTP URL，包含了标准输出以及错误日志（图 16.15）。每个作业 id 都可以被转换成多个 MapReduce 作业，取决于 Hive 或者 Pig 中运行中间阶段的数量。



Filename	Size	Last Modified
2012-11-24_161755_ekrishnan_script.g	0.1 kb	Sun, 25 Nov 2012 00:18:13 GMT
cmd.log	2.4 kb	Sun, 25 Nov 2012 00:18:36 GMT
conf/		Sun, 25 Nov 2012 00:18:18 GMT
hive logs/		Sun, 25 Nov 2012 00:18:22 GMT
stderr.log	0.2 kb	Sun, 25 Nov 2012 00:18:21 GMT
stdout.log	5.6 kb	Sun, 25 Nov 2012 00:18:21 GMT
tmp/		Sun, 25 Nov 2012 00:18:22 GMT

图 16.15 标准输出以及错误日志

6. Configuration Service

Configuration Service 被用于跟踪当前运行的集群以及支持的时间表。

当 Execution Service 接收到一个作业请求时，它通过 Configuration Service 将作业映射到合适的集群。如果存在多个满足作业需求的集群，它会随机选取一个集群。当然可以通过实现自定义负载均衡器来改进，以及分流单独的 Hadoop、Hive、Pig 作业，为每个作业分配独立的工作目录，从而实现 Genie 和作业本身的隔离。一个单独的 Genie 实例可以实现将不同集群提交的作业完全地从客户端中抽象出来。

16.5.3 Netflix 个性化和推荐系统架构

Netflix 的推荐和个性化功能向来精准。2014 年 3 月 27 日, Netflix 的工程师 Xavier Amatrain 和 Justin Basilico 在官方博客发布文章, 介绍了个性化和推荐系统架构。

如图 16.16 所示为推荐系统框架图, 其中的主要组件包括多种机器学习算法。

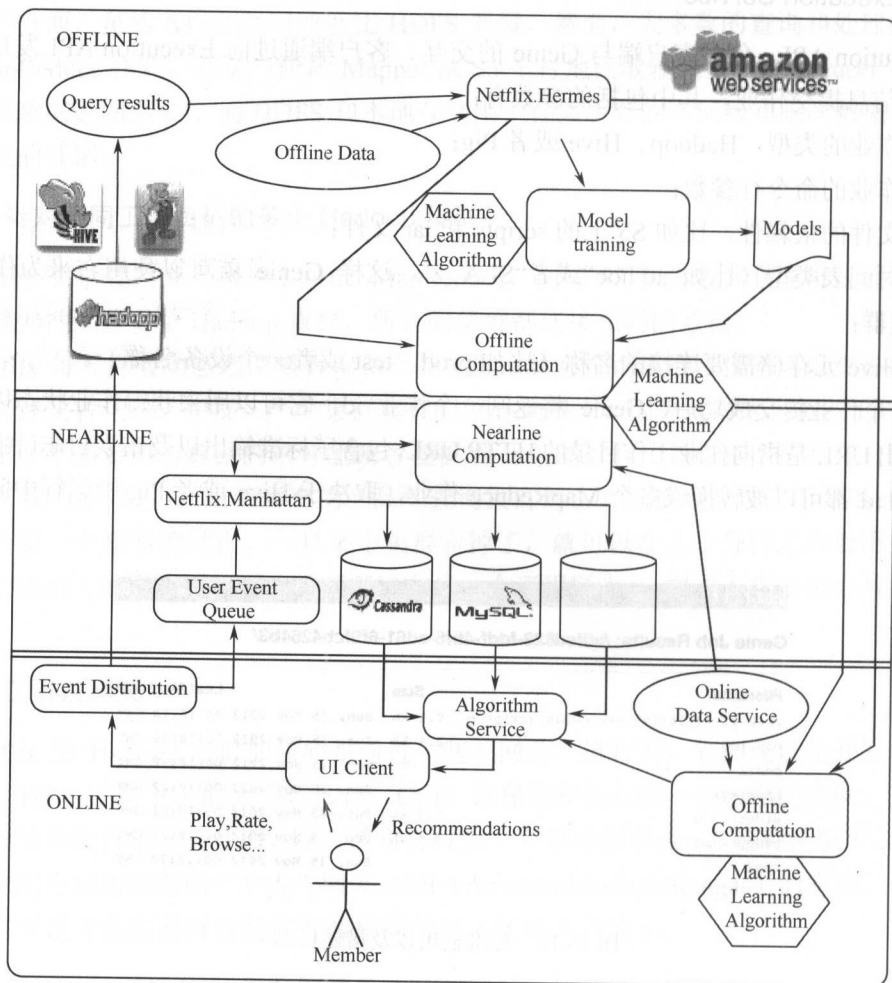


图 16.16 推荐系统框架图

1. 组件和处理过程

对于数据, 最简单的方法是存下来, 留作后续离线处理, 这就是用来管理离线作业 (Offline Job) 的部分架构。计算可以以离线、接近在线或在线方式完成。在线计算 (Online Computation) 能更快地响应最近的事件和用户交互, 但必须实时完成。这会限制使用算法

的复杂性和处理的数据量。离线计算（Offline Computation）对于数据数量和算法复杂度限制更少，因为它以批量方式完成，没有很强的时间要求。不过，由于没有及时加入最新的数据，所以很容易过时。个性化架构的关键问题，就是如何以无缝方式结合、管理在线和离线计算过程。接近在线计算（Nearline Computation）介于前两种方法之间，可以执行类似于在线计算的方法，但又不必以实时方式完成。模型训练（Model Training）是另一种计算，使用现有数据来产生模型，便于以后在实际结果计算中使用。另一部分架构是如何使用事件和数据分发（Event and Data Distribution）系统处理不同类型的数据和事件。与之相关的问题，是如何组合在离线、接近在线和在线之间跨越的不同信号和模型（Signal and Model）。最后，需要确定如何组合推荐结果（Recommendation Result），让其对用户有意义。

2. 在线、接近在线和离线计算

对于在线计算，相关组件需要满足 SLA 对可用性和响应时间的要求，而且纯粹的在线计算在某些情形下可能无法满足 SLA，因此，快速的备用方案就很重要，比如返回预先计算好的结果等。在线计算还需要不同的数据源确保在线可用，这需要额外的基础设施。

离线计算在算法上相对灵活，工程方面的需求也简单（图 16.17）。客户端的 SLA 响应时间要求也不高。在部署新算法到生产环境时，对于性能调优的需求也不高。Netflix 利用这种灵活性来完成快速实验：如果某个新的实验算法执行较慢，他们会部署更多 Amazon EC2 实例来达成吞吐处理目标，而不是花费宝贵的工程师时间去优化性能，因为业务价值可能不是很高。

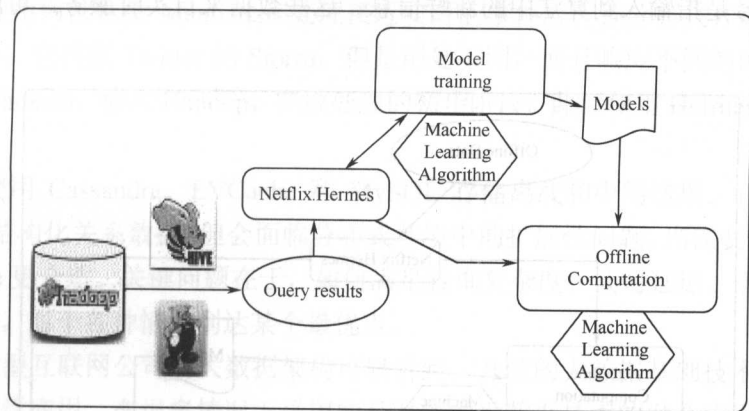


图 16.17 离线计算

接近在线计算与在线计算执行方式相同，但计算结果不是马上提供，而是暂时存储起来，使其具备异步性。接近在线计算的完成是为了响应用户事件，这样系统在请求之间响应速度更快。这样一来，针对每个事件就有可能完成更复杂的处理。增量学习算法很适合应用在接近在线计算中。

不管什么情况，选择在线、接近在线或离线处理，都不是非此即彼的决策。所有的方

对于事件和数据分发（图 16.19），Netflix 会从多种设备和应用中收集尽可能多的用户事件，并将其集中起来为算法提供基础数据。他们区分了数据和事件。事件是对时间敏感的信息，需要尽快处理。事件会路由、触发后续行动或流程。而数据需要处理和存储，便于以后使用，延迟并不重要，重要的是信息质量和数量。有些用户事件也会被作为数据处理。

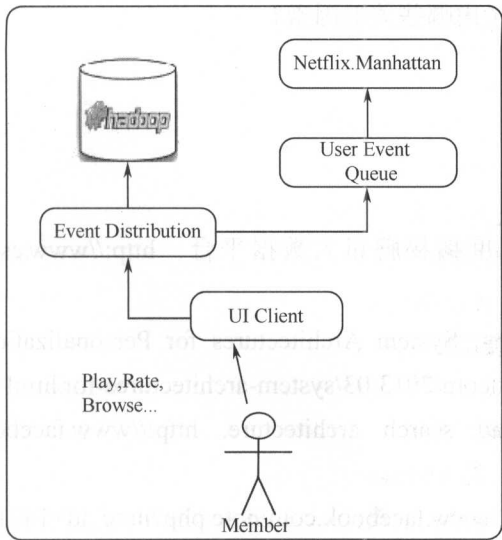


图 16.19 事件和数据分发

Netflix 使用内部框架 Manhattan 处理接近实时的事件流。该分布式计算系统是推荐算法架构的中心。它类似 Twitter 的 Storm，但是用处不同，而且响应不同的内部需求。数据流主要通过 Chukwa，输入 Hadoop，完成处理的初步阶段。此后使用 Hermes 作为发布—订阅机制。

Netflix 使用 Cassandra、EVCache 和 MySQL 存储离线和中间结果。它们各有利弊。MySQL 存储结构化关系数据，但会面临分布式环境中的扩展性问题。当需要大量写操作时，使用 EVCache 更合适。关键在于，如何满足查询复杂度、读写延迟、事务一致性等彼此冲突的需求，对于各种情况到达某个最优解。

从上述大型互联网公司的大数据架构可以看到，开源的大数据基础技术占据了主流，被这些公司广泛应用，在很多情况下采用的是对多种大数据技术的融合应用，实时计算技术也是各个公司架构中不可或缺的部分。同时也可发现，这些技术并不足以满足不同公司的不同业务需求，每一家企业都针对自己的业务特点做了定制和改进，很多技术又被回馈给开源社区。随着未来数据量的持续增长和对大数据处理的时效性、精准性、稳定性、扩展性的要求的进一步提高，将会涌现出更多的创新技术和实践，大数据技术将迈向新的台阶。

16.6 练习题

1. 总结大型系统的架构特点。
2. 设计系统架构要考虑哪些关键因素？

参考文献

- [1] 专访腾讯蒋杰：深度揭秘腾讯大数据平台. <http://www.csdn.net/article/2014-09-01/2821448>
- [2] The Netflix tech blog: System Architectures for Personalization and Recommendation. <http://techblog.netflix.com/2013/03/system-architectures-for.html>
- [3] Facebook's typeahead search architecture. <http://www.facebook.com/video/video.php?v=432864835468>
- [4] Facebook Chat. http://www.facebook.com/note.php?note_id=14218138919
- [5] Facebook's architecture presentation at Devoxx 2010. <http://www.devoxx.com>
- [6] Facebook 的系统架构. <http://smilejay.com/2012/09/architecture-of-facebook/>
- [7] 淘宝架构变迁. <http://www.oracle.com/technetwork/cn/community/developer-day/3-taobao-structure-457247-zhs.pdf>
- [8] <http://highscalability.com/blog/2013/7/8/the-architecture-twitter-uses-to-deal-with-150m-active-users.html>
- [9] Twitter 架构. <http://www.jdon.com/articheck/twitter.html>

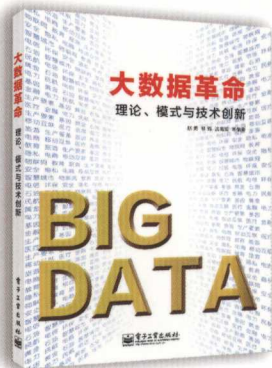
—大数据技术及算法解析 ARCHITECTING BIG DATA——TECHNOLOGIES AND ALGORITHMS EXPLAINED

主要内容

- **大数据技术综述及发展历程**：对大数据技术的起源以及近些年的发展趋势做了详细介绍。
- **大数据的技术分类**：包括基础架构支持，大数据采集，大数据存储，大数据处理，大数据展示及交互。
- **大数据行业的最新技术进展**：如Google新三驾马车，Spark统一计算平台，深度学习，可穿戴计算等。
- **大数据的发展趋势**：朝着实时化、泛在化、智能化的方向发展。
- **大型互联网公司的大数据架构实践**：包括腾讯、淘宝、Facebook、Twitter等。

本书可以作为大数据技术入门和进阶的专业书籍；
同时也可作为高等院校大数据相关课程的教材和教学参考书。

本书结合大量的大数据实践案例,对数据科学理论、大数据创新模式以及大数据技术作了探索 and 介绍,适合深入了解大数据技术
及应用的读者阅读。



ISBN 978-7-121-25978-4



定价：68.00元



策划编辑：董亚峰 (dyf@phei.com.cn)
责任编辑：董亚峰 (微信号：sundyf)
封面设计：朝天世纪